```
 GGG    PPPP      AAA
G   G   P   P    A   A
G       P   P    A   A
G  GG   PPPP     AAAAA
G   G   P        A   A
G   G   P        A   A
 GGGG   P        A   A
```

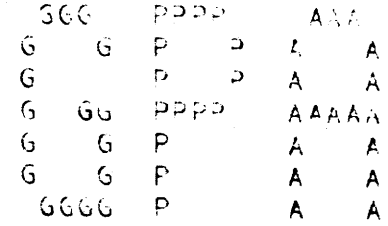
User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.OBJECT_MANAGEMENT
Version: V(5)
Request: 1266

Date: April 24, 1986
Queued: 11:25:43 AM
Printed: 11:34:13 AM

Volume II

```
-----------------
-- Export view --
-----------------

with Job;
with Disk;
with Error;
with Action;
with Naming;
with Object;
with Machine;
with Execution;
with Om_Services_1;
with Om_Definitions;
with Kernel_Services_1;

package Object_Management is
    -- This package defines the "standard export view" of the
    -- Object_Management subsystem (which for these purposes
    -- subsumes the Kernel functionality as well).

    -- Clients of this export view should always rename through this package.
    -- This will shield clients from the ravages of manual package integration,
    -- from a compilation point of view.

    -- Clients should ignore the following declarations:
    package P1 renames Job;
    package P2 renames Disk;
    package P3 renames Error;
    package P4 renames Action;
    package P5 renames Naming;
    package P6 renames Object;
    package P7 renames Machine;
    package P8 renames Om_Definitions;
    -- Clients should ignore the above declarations:

    ------------------------
    -- Exported packages --
    ------------------------

    package Job renames P1;
    package Disk renames P2;
    package Link renames Om_Services_1.Link;
    package User renames Om_Services_1.User;
    package View renames Om_Services_1.View;
    package Error renames P3;
    package World renames Om_Services_1.World;
    package Action renames P4;
    package Naming renames P5;
    package Object renames P6;
    package Machine renames P7;
    package Trigger renames Kernel_Services_1.Trigger;
    package Execution renames P7;
    package Wait_Service renames Kernel_Services_1.Wait_Service;
    package Access_Control renames Om_Services_1.Access_Control;
    package Om_Definitions renames P8;
    package Byte_String_Conversions renames
```

```
        Kernel_Services_1.Byte_String_Conversions;

end Object_Management;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.JOB
Version: V(63)
Request: 1267

Date: April 24, 1986
Queued: 11:25:51 AM
Printed: 11:34:1[illegible] AM

```
with Disk;
with Error;
with Machine;
with Calendar;
with Om_Definitions;

package Job is
    -- pragma Subsystem (Om_Definitions);

    package Om renames Om_Definitions;

    -- From the OM client's point of view, each command runs in a new job.

    -- This package attempts to capture most of the interesting "job state",
    -- as perceived by the OM client.  Job state includes:
    --     (a) The runtime representation of the "program".
    --     (b) The job world. This is a named world object containing the
    --         disk space which stores the runtime representation (i.e.,
    --         architectural modules and import spaces) of the "program".
    --     (c) Various job attributes, such as the job id, start time, etc.
    --     (d) A stack of various job state, such as current directory naming
    --         context.
    --     (e) A stack of views which controls version resolution by the
    --         Object package.
    --     (f) A stack of input/output/error files.
    --     (g) Scheduling information.
    --     (h) Persistently elaborated subsystems. The semantics are fully
    --         specified in the Execution package.

    -- Functions in this package attempt to return nil when errors are
    -- encountered.

    --------------------
    -- Job Attributes --
    --------------------

    function Exists (The_Job : Om.Job_Id) return Boolean;

    -- Return true iff there is a job with Current = The_Job.

    function Current return Om.Job_Id;

    function Start_Time (Of_Job : Om.Job_Id := Current) return Calendar.Time;
    function Elapsed_Time (Of_Job : Om.Job_Id := Current) return Duration;

    function Garbage_Directory return Om.Object_Handle;
    function Garbage_Directory (Of_Job : Om.Job_Id) return Om.Object_Handle;
    procedure Set_Garbage_Directory (Dir : Om.Object_Handle;
                                     Status : out Error.Condition;
                                     Of_Job : Om.Job_Id := Current);

    -- Initially, a job's garbage directory has the name
    --       !Machine.Job_Garbage.Unit_xxx.Job_yyy
    -- Where
    --     (1) "xxx" is the Unit_Number on which the job's world lives;
    --     (2) "yyy" is the Job_Number;
    --     (3) "unit_xxx" is the root of an object world;
    --     (4) "Job_yyy" is a directory;
```

```
   -- The job's garbage heaps live in this directory. Temp files created by
   -- Object.Create (with null name strings) go here. Rational created
   -- objects will follow the "_Foo_" convention to avoid conflicts with
   -- user applications, which are free to store objects in this directory.

   -- The system reserves the right to immediately expunge all objects in the
   -- directory (identified by the Garbage_Directory function) at job
   -- termination, but does not guarantee to do so. It is highly recommended
   -- that one never commit actions which create objects in the
   -- Garbage_Directory, since this policy will guarantee that job termination
   -- does not leave objects in the Garbage_Directory.

   -- The first function is equivalent to the second, supplying Job_Stack and
   -- Top.

   -- Possible errors:
   --    Is_Bad_Version_Handle (Status)
   --    Is_Access_Control_Error (Status)

   function Managed_Garbage_Heap return Machine.Segment_Name;
   function Managed_Garbage_Heap (Of_Job : Om.Job_Id)
                  return Machine.Segment_Name;

   -- This heap is created when the job starts, may be expunged when the job
   -- finishes, and is never "cut-back". Clients must manage their own
   -- garbage; that is, explicitly keep free lists. The first function
   -- is equivalent to the second, supplying Job.Current. This heap is created
   -- in the Garbage_Directory, with simple name "_Managed_Garbage_Heap_".

   function Stacked_Garbage_Heap return Machine.Segment_Name;
   function Stacked_Garbage_Heap (Of_Job : Om.Job_Id)
                  return Machine.Segment_Name;

   -- This heap is created when the job starts, may be expunged when the job
   -- finishes, and can be "cut-back" in a "stack-like" fashion. A program
   -- without tasking can use a technique where a subprogram can "mark" the
   -- stack on entry, and cut-back to the mark on exit.  The first function
   -- is equivalent to the second, supplying Job.Current.  This heap is created
   -- in the Garbage_Directory, with simple name "_Stacked_Garbage_Heap_".

   function Job_World return Om.Object_Handle;
   function Job_World (Of_Job : Om.Job_Id) return Om.Object_Handle;

   -- Returns the name of the directory object (a world) which contains the
   -- runtime representation of the identified job. This can be fed to the
   -- World package, which provides operations to get disk consumption
   -- statistics and operations to control disk space allocation.
   -- The first function is the same as the second if Current were supplied.

   ----------------------------------------
   -- Job Attributes, Session invariant --
   ----------------------------------------

   -- These operations return the same value for every job in the
   -- same session.

   function Session (Of_Job : Om.Job_Id := Current) return Om.Session_Id;
   -- Same for every job in the same "session".
```

```
   function User (Of_Job : Om.Job_Id := Current) return Om.Object_Handle;
   -- Object_Handle of root directory of user's world.
   -- Same for every job in the same "session".

   function User return Om.User_Id;
   function User (Of_Job : Om.Job_Id) return Om.User_Id;
   -- Internal, shorter, identity of user.

   function Account (Of_Job : Om.Job_Id := Current) return Om.Object_Handle;
   -- Object_Handle of a directory with subclass Account.
   -- Same for every job in the same "session".

   ------------------------------------
   --      Stack conventions         --
   ------------------------------------

   type Stack_Id is new Long_Integer;
   --  type Stack_Id (For_Session : Boolean) is
   --      record
   --          case For_Session is
   --              when false =>
   --                  The_Job : Om.Job_Id;
   --              when true =>
   --                  The_Session : Om.Session_Id;
   --          end case;
   --      end record;

   function Job_Stack (Of_Job : Om.Job_Id := Current) return Stack_Id;
   function Session_Stack (Of_Session : Om.Session_Id := Session)
               return Stack_Id;

   -- To operate on a stack associated with "your" job, simply supply the
   -- result of the Job_Stack function, with default parameter. To operate
   -- on a stack associated with "your" session, simply supply the result
   -- of the Session_Stack function, with default parameter. Note that
   --         Job_Stack (Root_Job (Session))
   -- and     Session_Stack (Current)
   -- are NOT the same!

   -- Subject to access control restrictions, one can operate on the stack
   -- of any job or session.

   subtype Stack_Location is Integer;
   Top : constant Stack_Location := 0;

   -- To look at the stack state that would be current following N
   -- invocations of the Pop_State operation, use a Stack_Location of Top-N.

   -- The "depth" of the stack is the number of "pops" that can be performed
   -- before getting stack underflow.

   ------------------------------------------------
   --        Job/Session state stack             --
   ------------------------------------------------

   function Stack_Depth (Stack : Stack_Id := Job_Stack) return Natural;
   function State_Exists (Stack : Stack_Id := Job_Stack;
```

```
                        Location : Stack_Location := Top) return Boolean;

    -- These can be used to determine the current depth of the stack, as well
    -- as the existance of particular stack locations.


    subtype Context is Om.Naming_Context;

    function Default_Context (Stack : Stack_Id := Job_Stack;
                              Location : Stack_Location := Top)
                 return Om.Pathname_String;
    function Default_Context return Context;
    function Default_Context (Stack : Stack_Id;
                              Location : Stack_Location) return Context;

    -- The second function is equivalent to the third, supplying Job_Stack and
    -- Top.

    procedure Set_Default_Context (The_Context : Om.Unique_Wildcard;
                                   Action : Om.Action_Id;
                                   Status : out Error.Condition;
                                   In_Context : Context := Default_Context;
                                   Stack : Stack_Id := Job_Stack;
                                   Location : Stack_Location := Top);

    -- The_Context, Action and In_Context are used according to the usual rules
    -- of Naming.Resolve to compute a Version_Handle, which is then stored as
    -- the default context for subsequent calls to Naming.Resolve.

    -- Possible errors:
    --    Is_Bad_Pathname (Status)
    --    Is_Bad_Action (Status)
    --    Is_Access_Control_Error (Status)

    procedure Set_Default_Context (The_Context : Context;
                                   Status : out Error.Condition;
                                   Stack : Stack_Id := Job_Stack;
                                   Location : Stack_Location := Top);

    -- Possible errors:
    --    Is_Bad_Version_Handle (Status)
    --    Is_Access_Control_Error (Status)


    function Name (Stack : Stack_Id := Job_Stack;
                   Location : Stack_Location := Top) return String;
    procedure Set_Name (New_Name : String;
                        Status : out Error.Condition;
                        Stack : Stack_Id := Job_Stack;
                        Location : Stack_Location := Top);

    -- Name of the job (as displayed by What.Jobs). Always the null string
    -- in the session stack.  Job names longer than about 100 characters
    -- are truncated at the left in order to fit within that limit.

    -- Possible errors:
    --    Is_Access_Control_Error (Status)
```

```
   function Termination_Message
                (Stack : Stack_Id := Job_Stack;
                 Location : Stack_Location := Top) return String;
   procedure Set_Termination_Message (New_Message : String;
                                      Status : out Error.Condition;
                                      Stack : Stack_Id := Job_Stack;
                                      Location : Stack_Location := Top);

   -- When non-null, job completion will include this string in message that
   -- goes to the initiator's message window.  This can be used by a program
   -- to display abnormal termination information.  Always the null string
   -- in the session stack.  Termination messages longer than about 1000
   -- characters are truncated in order to fit within that limit.

   -- Possible errors:
   --   Is_Access_Control_Error (Status)


   function Default_Wait return Duration;
   function Default_Wait (Stack : Stack_Id;
                          Location : Stack_Location) return Duration;
   procedure Set_Default_Wait (Status : out Error.Condition;
                               Default_Max_Wait : Duration := 5 * 60.0;
                               Stack : Stack_Id := Job_Stack;
                               Location : Stack_Location := Top);

   -- Used as the default time to wait for object locks.


   -- The first function is equivalent to the second, supplying Job_Stack and
   -- Top.

   -- Possible errors:
   --   Is_Access_Control_Error (Status)


   function Max_Actions return Integer;
   function Max_Actions (Stack : Stack_Id;
                         Location : Stack_Location) return Integer;
   procedure Set_Max_Actions (Status : out Error.Condition;
                              Max_Actions : Integer := 25;
                              Stack : Stack_Id := Job_Stack;
                              Location : Stack_Location := Top);

   function Max_Action_Locks return Integer;
   function Max_Action_Locks (Stack : Stack_Id;
                              Location : Stack_Location) return Integer;
   procedure Set_Max_Action_Locks (Status : out Error.Condition;
                                   Max_Action_Locks : Integer := 250;
                                   Stack : Stack_Id := Job_Stack;
                                   Location : Stack_Location := Top);

   -- Max_Actions serves to limit the number of actions which can be consumed
   -- by a job.  Currently, there is a limit of about 4000 concurrent actions.
   -- Since commands are compiled and therefore consume actions, running the
   -- system out of actions can be very painful! Each action lock consumes
   -- system resources as well.  Here the limit is the amount of disk space
```

```
    -- available.

    -- Possible errors:
    --   Is_Access_Control_Error (Status)


    package Profile is
        -- Currently a skeleton - to be filled in with "current spec"

        type Response_Profile is private;

        function Get (Stack : Stack_Id := Job_Stack;
                      Location : Stack_Location := Top)
                      return Response_Profile;
        procedure Set (New_Profile : Response_Profile;
                       Status : out Error.Condition;
                       Stack : Stack_Id := Job_Stack;
                       Location : Stack_Location := Top);

        -- Possible errors:
        --   Is_Access_Control_Error (Status)

        function Get_Default (Stack : Stack_Id := Job_Stack;
                              Location : Stack_Location := Top)
                      return Response_Profile;
        procedure Set_Default (New_Profile : Response_Profile;
                               Status : out Error.Condition;
                               Stack : Stack_Id := Job_Stack;
                               Location : Stack_Location := Top);

        -- Possible errors:
        --   Is_Access_Control_Error (Status)

    private
        type Response_Profile is new Boolean; -- ha ha
    end Profile;


    procedure Push_State (Status : out Error.Condition;
                          Stack : Stack_Id := Job_Stack);
    procedure Pop_State (Status : out Error.Condition;
                         Stack : Stack_Id := Job_Stack);

    -- Pop_State throws away the top element of the identified state stack.
    -- Push_State operation copies the top element into a new record, pushing
    -- it on top of the state stack.

    -- Possible errors:
    --   State_Stack_Overflow
    --   State_Stack_Underflow
    --   Is_Access_Control_Error (Status)

    ------------------------------------------
    --        Job/Session view stack        --
    ------------------------------------------

    -- This stack is operated independently of the state stack. It follows
    -- the same conventions. An additional feature: The System view appears
```

```
    -- to be an unremovable entry at the bottom of the stack.

    function View_Stack_Depth
                  (Stack : Stack_Id := Job_Stack) return Natural;
    function View_State_Exists
                  (Stack : Stack_Id := Job_Stack;
                   Location : Stack_Location := Top) return Boolean;

    -- These can be used to determine the current depth of the stack, as well
    -- as the existance of particular stack locations.


    function Default_View (Stack : Stack_Id := Job_Stack;
                           Location : Stack_Location := Top)
                  return Om.Pathname_String;
    function Default_View (Stack : Stack_Id := Job_Stack;
                           Location : Stack_Location := Top) return Context;

    procedure Set_Default_View (The_Context : Om.Unique_Wildcard;
                                Action : Om.Action_Id;
                                Status : out Error.Condition;
                                In_Context : Context := Default_Context;
                                Stack : Stack_Id := Job_Stack;
                                Location : Stack_Location := Top);

    -- Possible errors:
    --    Is_Bad_Pathname (Status)
    --    Is_Bad_Action (Status)
    --    Is_Access_Control_Error (Status)

    procedure Set_Default_View (The_Context : Context;
                                Status : out Error.Condition;
                                Stack : Stack_Id := Job_Stack;
                                Location : Stack_Location := Top);

    -- These operations simply read/write the identified location in the
    -- identified view stack. Note that if the stack is empty, the functions
    -- will return the System view.

    -- Possible errors:
    --    Is_Bad_Version_Handle (Status)
    --    Is_Access_Control_Error (Status)


    function Default_World_View (Of_World : Machine.Object_Worlds)
                  return Om.Version_Id;
    function Default_World_View (Of_World : Machine.Object_Worlds;
                                 Stack : Stack_Id) return Om.Version_Id;

    -- Scans down the identified universe view stack looking for a universe
    -- view which has a non-nil slot for the given world. For this purpose,
    -- treats the stack as if the System view is at the bottom. Returns the
    -- Version_Id of the selected universe view. If the stack does not contain
    -- such a universe view, returns Nil. This operation does NOT follow the
    -- indirect chain which may lead from the slot of the returned universe
    -- view. Note that a non-nil result does not imply that the referenced
    -- universe view selects a world view which exists (in particular, an
    -- indirect chain may be broken).
```

```
    -- This operation is used by the Object package to bind Version_Handle's.

    -- This first function is equivalent to the second, when given Job_Stack.

    procedure Push_View_State (Status : out Error.Condition;
                               Stack : Stack_Id := Job_Stack);
    procedure Pop_View_State  (Status : out Error.Condition;
                               Stack : Stack_Id := Job_Stack);

    -- Pop_Universe_State throws away the top element of the identified view
    -- stack. Push_Universe_State operation copies the top element into a new
    -- record, pushing it on top of the identified view stack.

    -- Possible errors:
    --   View_Stack_Overflow
    --   View_Stack_Underflow
    --   Is_Access_Control_Error (Status)

    -----------------
    -- Job's files --
    -----------------

    -- Note that current input/output/error have their own independent job
    -- state stacks. Job start and finish semantics for job files is
    -- implemented using a coupling mechanism which is not generally available.
    -- See the Job_Files package in DIO.

    ----------------------
    -- Execution context --
    ----------------------

    -- Each job's session has an execution state stack which follows the same
    -- basic paradigm as the job view stack. See the Execution package for
    -- details.

    --------------------
    -- Job scheduling --
    --------------------

    package Scheduling is

        function Kind (Of_Job : Om.Job_Id)
                       return Om.Job_Kind;

        -- Returns Om.Terminated for jobs which do not "exist".

        function Priority (Of_Job : Om.Job_Id := Job.Current)
                       return Om.Job_Priority;
        function State (Of_Job : Om.Job_Id := Job.Current)
                       return Om.Job_State;
        function Cpu_Consumption (Of_Job : Om.Job_Id := Job.Current)
                       return Long_Integer; -- msec
        function Disk_Consumption (Of_Job : Om.Job_Id := Job.Current)
                       return Long_Integer; -- blocks transferred
        function Current_Memory_Consumption
                       (Of_Job : Om.Job_Id := Job.Current)
```

```
                    return Long_Integer; -- pages in cache

    -- Returns Nil for non existant jobs.

    procedure Enable (The_Job : Om.Job_Id;
                      Status : out Error.Condition);
    procedure Disable (The_Job : Om.Job_Id;
                       Status : out Error.Condition);

    -- Possible errors:
    --   Is_Access_Control_Error (Status)

    function Working_Set_Limit (Of_Job : Om.Job_Id := Job.Current)
                  return Long_Integer; -- cache pages
    procedure Set_Working_Set_Limit
                  (The_Job : Om.Job_Id := Job.Current;
                   New_Limit : Long_Integer;   -- cache pages
                   Status : out Error.Condition);

    -- Possible errors:
    --   Is_Access_Control_Error (Status)

    procedure Attach (Job : Om.Job_Id;
                      Status : out Error.Condition);
    procedure Detach (The_Job : Om.Job_Id;
                      Status : out Error.Condition);

    -- Possible errors:
    --   Is_Access_Control_Error (Status)

    function Run_Load (Interval : Duration := 60.0) -- seconds
                  return Long_Integer; -- load * 100

    function Disk_Load (Interval : Duration := 60.0) -- seconds
                  return Long_Integer; -- load * 100

    function Withheld_Load (Interval : Duration := 60.0) -- seconds
                  return Long_Integer; -- load * 100


    -- more stuff to follow
end Scheduling;

---------------------
-- Session control --
---------------------

function Exists (The_Session : Om.Session_Id) return Boolean;

-- Returns true iff there exists at least one job for which the Session
-- function matches given Session_Id parameter.

function Default_Unit (Of_Session : Om.Session_Id := Job.Session)
                return Disk.Unit_Number;
procedure Set_Default_Unit (Of_Session : Om.Session_Id := Job.Session;
                            Unit : Disk.Unit_Number;
                            Status : out Error.Condition);
```

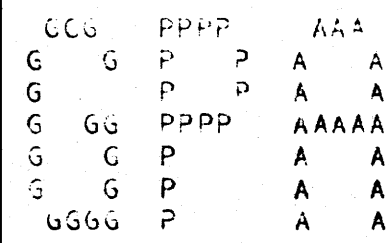
```
    -- A value of Nil means that the system will choose the "best" unit
    -- on which to run new jobs for the given session. A non-nil value
    -- causes the system to always run new jobs on the specified unit (unless
    -- extenuating circumstances require otherwise).

    -- Possible errors :
    --   Is_Access_Control_Error (Status)

    function Root_Job (Of_Session : Om.Session_Id := Job.Session)
                  return Om.Job_Id;

    -- Returns the id of the first job created under the given session. If the
    -- session parameter is garbage, or the job does not exist, returns Nil.

    function Subsystem_State (Of_Session : Om.Session_Id := Job.Session)
                  return Om.Job_Id;

    -- A session may have an associated job which is used to keep persistently
    -- elaborated copies of subsystems. This function returns Nil when the
    -- session does not exist or does not have such an associated job. See
    -- the Execution package for further details.

    procedure Kill_Session (The_Session : Om.Session_Id;
                            Status : out Error.Condition);

    -- Does a Kill on every job with the given Session_Id.

    -- Possible errors:
    --   includes those for Job.Kill on a Job_Id

    procedure wait_For_Termination (The_Session : Om.Session_Id);

    -- returns immediately when The_session is garbage or otherwise does not
    -- exist.

    function Terminal (Of_Session : Om.Session_Id := Job.Session)
                  return Om.Version_Handle;

    -- Identifies the terminal to which the session is "attached",
    -- possible Nil.

    function Message_Window (Of_Session : Om.Session_Id := Job.Session)
                  return Om.Version_Handle;
    function Io_Window (Of_Session : Om.Session_Id := Job.Session)
                  return Om.Version_Handle;

    -- Return files with subclass Message_window and Io_Window, respectively.
    -- Used with DIO to read/write to screen via editor.

    ------------------------------------
    -- Job/Session creation/deletion --
    ------------------------------------

    -- There is no relationship between jobs other than the fact some jobs have
    -- the same Session_Id. In particular, there is no notion of parent/child
    -- etc.

    generic
```

```
        type Parameters is private;
        with procedure Execute (P : in out Parameters);
    package Initiate_Generic is

        procedure Initiate
                        (Name : String;
                         P : in out Parameters;
                         New_Job : out Om.Job_Id;
                         Status : out Error.Condition;

                         -- Initial job files:
                         Current_Output : Om.Unique_Wildcard := "";
                         Current_Error : Om.Unique_Wildcard := "";
                         Current_Input : Om.Unique_Wildcard := "";

                         -- New session parameters:
                         Start_New_Session : Boolean := False;
                         New_User : Om.Object_Handle := Om.Nil_Object_Handle;
                         Users_Password : String := "";
                         Users_Account : String := "";

                         Terminal : Om.Object_Handle := Om.Nil_Object_Handle;

                         -- Miscellaneous parameters:
                         Unit : Disk.Unit_Number := Job.Default_Unit;
                         After : Duration := 0.0;
                         Job_Kind : Om.Job_Kind := Om.Attached_Job;
                         Wait_For_Completion : Boolean := False);

        -- The job's world will live on the specified unit (or "best" unit
        -- when Nil is specified).

        -- The first architectural module created in a new job is known as the
        -- "root thread"; there is a function to fetch this value. The
        -- generic formal subprogram (Execute) is run on the root thread of
        -- the new job. The job terminates when this root thread terminates.

        -- The first job created in a new session is known as the "root job";
        -- there is a function to fetch this value. For interactive
        -- sessions, this job runs the editor.

        -- Setting Start_New_Session to true is used to run a job under a
        -- different Session_Id.  If the value of New_User is Nil, the new
        -- session runs with the id of Job.User, and no password check is
        -- performed. Otherwise, checks password. Checks other login profile
        -- parameters. Updates time of last login; when the last job of the
        -- session terminates, will update time of last logout; also appends
        -- appropriate entries to the accounting log.

        -- Caller can implement almost arbitrary rules for the state of the
        -- new job by passing information via the Parameters to the Execute
        -- procedure (and feeding these values to the various job state Set_
        -- operations, above).

        -- The built in rules for inheritance of the state and view stack
        -- are as follows:
        --      (1) when starting a new session, both the new session and its
        --          root job have state and view stacks which are copies of the
```

```
      --          initiator's job stacks. This allows the session initiator
      --          to use its job stacks as parameters to this operation.
      --      (2) when starting a new job (within an already existing
      --          session), the state stack, view stack and execution state
      --          stack of the new job are initially copies of the
      --          corresponding session stacks.
      -- Note that "disconnect" simply detaches an already running job,
      -- leaving it with whatever job stacks it had. The new job (created to
      -- run commands) will start with session state (as defined in rule 2
      -- above). Thus, jobs share changing state only when they explicitly
      -- manipulate session state.

      -- In any case, a Set_Name (and Push_State if necessary) is
      -- performed to set the job name as specified.

      -- The io files for the new job are always as specified by the
      -- job files parameters. Recall that setting an io file to "" will
      -- cause first reference to the file to attempt to open the
      -- session's terminal.

      -- The persistent subsystem state of sessions & jobs is specified
      -- in the Execution package.

      -- Job termination will send a non-Nil Termination_Message to the job's
      -- current error file. Recall that "termination" includes normal
      -- job completion as well as operations such as Job.Terminate_Self
      -- and Job.Kill.

      -- Possible errors:
      --   Is_Resource_Limit_Error (Status)
      --       o System_Is_Out_Of_Job_World_Numbers
      --   Is_Bad_Start_Job_Parameter (Status)
      --       o Bad_Job_Current_Output_Parameter
      --       o Bad_Job_Current_Error_Parameter
      --       o Bad_Job_Current_Input_Parameter
      --   Is_Login_Error (Status)
      --       o Login_User_Id_Does_Not_Exist
      --       o Login_User_Password_Invalid

   end Initiate_Generic;


   procedure Kill (The_Job : Om.Job_Id;
                   Status : out Error.Condition;
                   Message : String := "");

   -- Works on self and editors. If the caller supplies a non-null Message,
   -- its value is assigned to the killed job's Termination_Message.
   -- Otherwise, the job's Termination_Message will be automatically set to
   -- either
   --      (1)  "<job> has been killed"   or
   --      (2)  "<job> has been killed by <user>"
   -- where the value of <job> comes from the killed job's Name, and the
   -- value of <user> comes from the User function of the executing job.
   -- The first string is produced when the session of the killer is the same
   -- as the session of the killed job. The second string is produced in the
   -- remaining cases.
```

```
   -- Possible errors:
   --    Is_Execution_Error (Status):
   --        o Subsystem_Is_Still_Shared
   --            The specified job has elaborated copies of subsystem which
   --            are currently shared by other jobs.
   --    Is_Access_Control_Error (Status);

   procedure Terminate_Self;

   -- Same as Job.Kill on self, but produces no job termination message.

   procedure wait_For_Termination (The_Job : Om.Job_Id);

   -- Returns immediately when The_Job is garbage or otherwise does not exist.

   ----------------------
   -- Tasks within job --
   ----------------------

   function My_Task_Id return Machine.Task_Id;

   -- "task id" of the executing thread. Note that packages also have
   -- task ids.

   function Job_Of (The_Task : Machine.Task_Id) return Om.Job_Id;

   -- The job to which the task belongs. When this job terminates, the task
   -- is certainly terminated.

   function Root_Thread (Of_Job : Om.Job_Id := Current) return Machine.Task_Id;

   -- The "task id" of the first thread created in the job.  If this task
   -- dies, the job will certainly be dealt a severe blow (its actions would
   -- be abandoned, for example). Other than some interesting issues related
   -- to allocation of tasks from persistently elaborated collections, the
   -- job would probably eventually die.

   function Is_Callable (The_Task : Machine.Task_Id) return Boolean;

   -- As per the 'Callable attribute in the LRM. Used by the action manager
   -- to determine if an action's guardian is "alive" (not alive causing the
   -- action to be auto abandoned).

   procedure Set_Associated_Heap (To_Heap : Machine.Segment_Name);
   function Get_Associated_Heap return Machine.Segment_Name;
   function Get_Associated_Heap (For_Module : Machine.Module_Name)
               return Machine.Segment_Name;

   -- The above operations are used by Diana to set/get the task's current
   -- Diana unit.  The second Get_ operation is subject to access control.

   generic
       type Task_Type is limited private;
       type Pointer is access Task_Type;

   procedure Allocated_New_Root_Thread
        (In_Job : Om.Job_Id := Job.Current;
         New_Task : out Pointer;
```

```
         Status : out Error.Condition);

   -- This operation allows one to allocate a task in another job. The new
   -- task is considered to be an additional root thread in the target job.
   -- Termination of the target job will cause the termination of all these
   -- additional root threads.

   -- In the Gamma system the rules for task allocation (in a collection) are
   -- that the allocated task runs in the same job as the allocating thread
   -- (unless pragma Virtual_Processor is used). In Epsilon, the pragma
   -- Virtual_Processor is not available, and tasks cannot be allocated in
   -- other jobs without using this operation.

   -- Possible errors:
   --   Is_Access_Control_Error (Status);

end Job;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.KK.DISK
Version: V(30)
Request: 1268

Date: April 24, 1986
Queued: 11:25:58 AM
Printed: 11:35:20 AM

```
with Sys;
with Machine;
with Um_Definitions;

package Disk is
    package System renames Sys;

    Nil : constant := 0;

    -- Null value for all scalar types in this package,
    -- unless stated otherwise.

    type Unit_Number is new Long_Integer range 0..15;

    -- Denotes the physical drive unit number.  Note that the unit number of
    -- a drive can change when its jumpers are changed.

    type Spindle_Size is
        record
            Cylinders : Long_Integer range 0..2 ** 16 - 1;
            Tracks    : Long_Integer range 0..2 ** 8 - 1;
            Sectors   : Long_Integer range 0..2 ** 8 - 1;
        end record;

    pragma Assert (Spindle_Size'Size = 32);

    type Sector_Number is new Long_Integer range 0..2 ** 8 - 1;
    type Track_Number is new Long_Integer range 0..2 ** 8 - 1;
    type Cylinder_Number is new Long_Integer range 0..2 ** 12 - 1;

    type Address is new Long_Integer range 0..2 ** 28 - 1;
    --    record
    --        Cylinder : Cylinder_Number;
    --        Track    : Track_Number;
    --        Sector   : Sector_Number;
    --    end record;
    -- Corresponds to the disk address at the IOP interface

    Can_Pad_Address_To_Get_Iop_Address : constant Boolean := True;
    -- Can pad Address on the left with zeroes, and treat it as a subtype
    -- of the iop's disk address format.

    function To_Address
                  (Cylinder : Cylinder_Number;
                   Track : Track_Number;
                   Sector : Sector_Number) return Address;
    function Cylinder (Addr : Address) return Cylinder_Number;
    function Track (Addr : Address) return Track_Number;
    function Sector (Addr : Address) return Sector_Number;

    function Increment (Addr : Address; Size : Spindle_Size) return Address;
    function Decrement (Addr : Address; Size : Spindle_Size) return Address;

    -- Return the addresses of the next and prior blocks, respectively, of
    -- the given address.

    type Address_Array is array (Natural range <>) of Address;
```

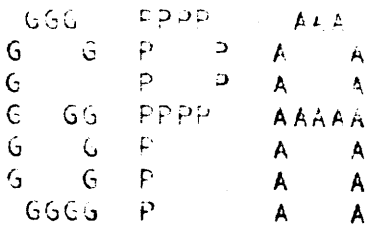
```
   type Address_Range is new Long_Integer range 0..2 ** 56 - 1;
   --    record
   --        First : Address;
   --        Last  : Address;
   --    end record;

   function To_Address_Range
                (First : Address; Last : Address) return Address_Range;
   function First (The_Range : Address_Range) return Address;
   function Last (The_Range : Address_Range) return Address;

   Null_Address_Range : constant Address_Range := To_Address_Range (1, 0);

   function Is_Null_Range (The_Range : Address_Range) return Boolean;
   function In_Range (Addr : Address; The_Range : Address_Range)
                return Boolean;
   function Ranges_Overlap (Range_1, Range_2 : Address_Range) return Boolean;

   type Address_Range_Array is array (Natural range <>) of Address_Range;

   Sector_Size_In_Bytes : constant := 512;

   subtype Region_Number is Cylinder_Number;

   -- A region is composed of one or more adjacent disk cylinders, and is
   -- denoted by the Cylinder_Number of the first cylinder in the region.

   subtype Region_Count is Long_Integer range 0..2 ** 16 - 1;
   pragma Assert (Region_Count'Size >= Cylinder_Number'Size);

   subtype Block_Address is Address;

   -- A block is composed of one or more adjacent sectors.

   subtype Block_Count is Long_Integer range 0..2 ** 32 - 1;
   pragma Assert (Block_Count'Size >= Address'Size);

   Block_Size_Equals_Page_Size : constant Boolean := True;
   Blocks_Span_Tracks          : constant Boolean := False;

   subtype Cluster_Address is Address;

   -- A cluster is composed of one or more adjacent blocks.  The cluster
   -- size may vary from world to world.  Note that the Volume_Manager
   -- allocates clusters, not blocks.

   Cluster_Size_Is_Multiple_Of_Block_Size : constant Boolean := True;
   Clusters_Span_Tracks : constant Boolean := False;

   Cluster_Size_Equals_Page_Size          : constant Boolean := True;
   -- True in the first implementation, but false in future revs.

   subtype Cluster_Count is Long_Integer range 0..2 ** 32 - 1;
   pragma Assert (Cluster_Count'Size >= Address'Size);

   type Full_Address is new Long_Integer range 0..2 ** 32 - 1;
   --    record
   --        Unit     : Unit_Number;
```

```
--          Cylinder : Cylinder_Number;
--          Track    : Track_Number;
--          Sector   : Sector_Number;
--      end record;

    function To_Full_Address
                 (Unit : Unit_Number;
                  Cylinder : Cylinder_Number;
                  Track : Track_Number;
                  Sector : Sector_Number) return Full_Address;
    function To_Full_Address
                 (Unit : Unit_Number; Addr : Address) return Full_Address;
    function To_Address (Addr : Full_Address) return Address;
    function Unit (Addr : Full_Address) return Unit_Number;
    function Cylinder (Addr : Full_Address) return Cylinder_Number;
    function Track (Addr : Full_Address) return Track_Number;
    function Sector (Addr : Full_Address) return Sector_Number;

    function Increment (Addr : Full_Address;
                        Size : Spindle_Size) return Full_Address;
    function Decrement (Addr : Full_Address;
                        Size : Spindle_Size) return Full_Address;

    -- Return the addresses of the next and prior blocks, respectively, of
    -- the given address.

    type Full_Address_Array is array (Natural range <>) of Full_Address;

    subtype Full_Block_Address is Full_Address;
    subtype Full_Cluster_Address is Full_Address;

    All_Dirty_World_Info_Mapped_Under_Its_Vp : constant Boolean := True;

    -- This invariant is fundamental to the window of vulnerability mechanism,
    -- actions, and crash recovery. Therefore, one should take great care
    -- with the world parameter in the following function.

    -- Be extra careful when using "special worlds" (like Machine.Om_World and
    -- Machine.Miscellaneous_World).

    function To_Disk_Mapping_Address
                 (Addr : Full_Address;
                  World : Machine.World_Number) return Machine.Page_Address;

    -- All unwired kk_om data structures are in cache pages whose vp
    -- corresponds to the world in which the data structure lives.  Data
    -- structures which are not implemented in heaps use this function to map
    -- their disk addresses to cache pages.

    -- Converts the disk address to a page address as follows:
    --     0                5 bits  \
    --     unit             4 bits   --> segment
    --     cylinder        12 bits  /
    --     world param     10 bits  ---> vp
    --     track            6 bits  \
    --     sector           8 bits   --> page number
    --     0                3 bits  /
    --     0               10 bits  ---> bit offset
```

```
--    4                3 bits  ---> segment kind (data)
--                   -----------
--                     64 bits

-- Note that this mapping implies that ucode must not allocate module
-- names with segment numbers smaller than 2 ** 16 (ie, 1/64 of the
-- name space is preallocated by OM).

-- Note that it is possible for the same Full_Address to be mapped by many
-- different Page_Addresses, each with a different World_Number component.
-- By convention, DON'T DO THAT!

function Is_Disk_Mapping_Page (Addr : Machine.Page_Address) return Boolean;

-- Returns true iff the most significant 6 bits of the address are 0 and
-- the least significatnt bits of the address are 4 (kind).

function To_Disk_Address (Addr : Machine.Page_Address) return Full_Address;

-- Inverts the above mapping. Returns a bogus disk address if the given
-- page address is not a disk mapping page (the bogus result is guaranteed
-- to cause an error when used to issue a physical disk IO).

type Region_Id is new Long_Integer range 0..2 ** 16 - 1;
--   record
--       Unit   : Unit_Number;
--       Region : Region_Number;
--   end record;

function To_Region_Id (Unit : Unit_Number;
                       Region : Region_Number) return Region_Id;
function Unit (Id : Region_Id) return Unit_Number;
function Region (Id : Region_Id) return Region_Number;

function Base_Address (Of_Region : Region_Id) return Address;
function Full_Base_Address (Of_Region : Region_Id) return Full_Address;
-- Returns an address with same cylinder #, and zero for track and sector.

type Block_Kinds is new Long_Integer range 0..2 ** 8 - 1;

-- "Standard header" for disk blocks:
--   record
--       Kind     : Block_Kinds;
--       ...      : Filler.Filler_8;
--       Instance : Om_Definitions.Unique_Id;
--       Home     : Full_Address;
--       -- the above fields consume the first 96 bits.
--       ...
-- "Semi-standard header" for disk blocks:
--   record
--       ...      : Filler.Filler_64;
--       ...      : Filler.Filler_64;
--       Kind     : Block_Kinds;
--       ...      : Filler.Filler_8;
--       Instance : Om_Definitions.Unique_Id;
--       Home     : Full_Address;
--       -- the above fields consume the first 128 + 96 bits.
--       ...
```

```
   -- This convention is fundamental to the ability to detect garbage in
   -- kernel data structures.  It also greatly simplifies one's ability to
   -- poke around at disk data structures, and make sense out of them.
   -- The Pack_Label has a non-standard header, for historical reasons.
   -- The List_Section_Block and Object_Header_Block have semi-standard
   -- headers for compatability with ucode interpretation of the first
   -- word in a heap.  Unless stated otherwise, all other block kinds
   -- have standard headers.  Block_Kinds takes on the following values:

   Size_Of_The_Standard_Block_Header : constant :=
       Block_Kinds'Size + 8 +
       Um_Definitions.Unique_Id'Size +
       Full_Address'Size;

   Uninitialized_Block               : constant Block_Kinds := 16#0#;
   -- Block_Kind has not yet been assigned to the block.


   Retarget_Database_Block           : constant Block_Kinds := 16#1#;
   -- Individual block in the retarget database.

   World_Allocation_Block            : constant Block_Kinds := 16#2#;
   -- Map from world_Number to location of world_Header_Block.

   World_Header_Block                : constant Block_Kinds := 16#3#;
   -- At the beginning of the root region of an existing world.

   List_Section_Block                : constant Block_Kinds := 16#4#;
   -- Used to construct lists of disk addresses.
   -- Uses "semi-standard header".

   Object_Header_Block               : constant Block_Kinds := 16#5#;
   -- Contains ucode control, KK_OS control, and initial user data of heap.
   -- Uses "semi-standard header".

   Free_Object_Header_Block          : constant Block_Kinds := 16#6#;
   -- An unused object header block.
   -- Uses "semi-standard header".

   Machine_Segment_Index_Block       : constant Block_Kinds := 16#10#;
   -- Index_Block for a segment created by the architecture.

   Heap_Segment_Index_Block          : constant Block_Kinds := 16#11#;
   -- Index_Block for a segment underlying a permanent object.

   Object_Catalog_Branch_Block       : constant Block_Kinds := 16#20#;
   Object_Catalog_Leaf_Block         : constant Block_Kinds := 16#21#;
   Job_Catalog_Branch_Block          : constant Block_Kinds := 16#22#;
   Job_Catalog_Leaf_Block            : constant Block_Kinds := 16#23#;
   -- Btree nodes for the segment catalogs.

   Structural_Map_Branch_Block       : constant Block_Kinds := 16#30#;
   Structural_Map_Leaf_Block         : constant Block_Kinds := 16#31#;
   -- Btree nodes for directory's structural map.

   Version_Map_Branch_Block          : constant Block_Kinds := 16#32#;
   Version_Map_Leaf_Block            : constant Block_Kinds := 16#33#;
   -- Btree nodes for directory's version map.
```

```
    Version_Map_Data_Block              : constant Block_Kinds := 16#34#;
    -- Contains information about particular versions of objects.

    Link_Pack_Branch_Block_1            : constant Block_Kinds := 16#36#;
    Link_Pack_Leaf_Block_1              : constant Block_Kinds := 16#37#;
    -- Btree nodes for the reconstructed portion of the Link_Pack state.

    Link_Pack_Branch_Block_2            : constant Block_Kinds := 16#38#;
    Link_Pack_Leaf_Block_2              : constant Block_Kinds := 16#39#;
    -- Btree nodes for the temporary portion of the Link_Pack state.

    Action_Log_Head_Block               : constant Block_Kinds := 16#40#;
    Action_Log_Block : constant Block_Kinds := 16#41#;
    -- Used to build action logs.


end Disk;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.KK.ERROR
Version: V(51)
Request: 1269

Date: April 24, 1986
Queued: 11:26:05 AM
Printed: 11:35:44 AM

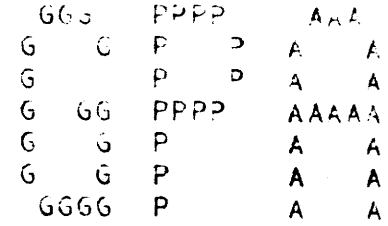
```
package Error is

    type Status is new Long_Integer range 0..2 ** 16 - 1;
    -- Vanilla status code

    type Condition_Information is new Long_Integer range 0..2 ** 48 - 1;

    type Condition is new Long_Integer;
    -- Status plus info which is a function of the status
    -- record
    --    Info   : Condition_Information;
    --    Reason : Status;
    -- end record;

    function Info (The_Condition : Condition) return Condition_Information;
    function Reason (The_Condition : Condition) return Status;
    function To_Condition (Info : Condition_Information;
                           Reason : Status) return Condition;

    -------------
    -- Success --
    -------------

    Successful_Status : constant Status := 0;
    Successful        : constant Condition := 0;

    function Is_Successful (The_Status : Status) return Boolean;
    function Is_Successful (The_Condition : Condition) return Boolean;

    -- Returns true for the following values of Status:
    --   Successful

    -- It is considered OK for clients to use "=" to compare either a
    -- Status or Condition to the value Successful, instead of calling the
    -- Is_Successful function, which cannot be inlined due to compiler/debugger
    -- restrictions.

    ---------------------------------------
    -- Bad Object_Id/Version_Id (010x) --
    ---------------------------------------

    function Is_Bad_Version_Id (The_Status : Status) return Boolean;
    function Is_Bad_Version_Id (The_Condition : Condition) return Boolean;
    function Is_Bad_Object_Id (The_Status : Status) return Boolean;
    function Is_Bad_Object_Id (The_Condition : Condition) return Boolean;
    function Is_Bad_Version_Handle (The_Condition : Condition) return Boolean;
    function Is_Bad_Object_Handle (The_Condition : Condition) return Boolean;

    World_Does_Not_Exist            : constant Status := 16#0101#;
    Object_Class_Does_Not_Exist     : constant Status := 16#0102#;
    Object_Index_Does_Not_Exist     : constant Status := 16#0103#;
    Object_Instance_Does_Not_Match  : constant Status := 16#0104#;
    Object_Version_Does_Not_Exist   : constant Status := 16#0105#;
    No_World_View : constant Status := 16#0106#;
    Object_Not_In_View              : constant Status := 16#0107#;
    Universe_Does_Not_Exist         : constant Status := 16#0108#;
    Max_Indirection_Exceeded        : constant Status := 16#0109#;
```

```
   -------------------------
   -- Bad pathname (011x) --
   -------------------------

   function Is_Bad_Pathname (The_Status : Status) return Boolean;
   function Is_Bad_Pathname (The_Condition : Condition) return Boolean;

   Filename_Syntax_Error           : constant Status := 16#0111#;
   --    Condition identifies the character in the name which caused the
   --    error, as well as the token class which was expected.
   Child_Object_Does_Not_Exist     : constant Status := 16#0112#;
   --    Condition identifies the segment (of the name) which caused the
   --    error.  The prefix to the left of this segment resolved without
   --    error.  The identified segment does not identify any child object.
   Link_Pack_Entry_Does_Not_Exist : constant Status := 16#0113#;
   --    Condition identifies a segment (of the name) which was resolved by
   --    looking through the link pac, which failed.
   Child_Name_Is_Ambiguous         : constant Status := 16#0114#;
   --    Similar to the previous status, but the identified name segment
   --    identifies more than one child object.
   Set_Causes_Ambiguity            : constant Status := 16#0115#;
   --    Condition identifies a segment (of the name) which uses set
   --    notation to identify more than one object/version.
   Indirect_File_Causes_Ambiguity : constant Status := 16#0116#;
   --    Condition identifies a segment (of the name) which identifies an
   --    indirect file which identifies more than one object/version.

   ------------------------
   -- Bad action (012x) --
   ------------------------

   function Is_Bad_Action (The_Status : Status) return Boolean;
   function Is_Bad_Action (The_Condition : Condition) return Boolean;

   Action_Does_Not_Exist            : constant Status := 16#0121#;
   Not_The_Action_Owner             : constant Status := 16#0122#;
   Commit_Has_Been_Prevented        : constant Status := 16#0123#;
   Action_Already_Has_Update_Locks : constant Status := 16#0124#;
   Update_Has_Been_Prevented        : constant Status := 16#0125#;
   Action_Owner_Not_Willing         : constant Status := 16#0126#;
   Cant_Xfer_Ownership_Cross_Jobs   : constant Status := 16#0127#;
   Action_Owner_Not_Dead            : constant Status := 16#0128#;
   Master_Action_Does_Not_Exist     : constant Status := 16#0129#;
   Action_Is_A_Dependent_Action     : constant Status := 16#012A#;
   Action_Already_Has_Dependent     : constant Status := 16#012B#;
   Action_Guardian_Not_Dead         : constant Status := 16#012C#;

   ------------------------
   -- Lock errors (013x) --
   ------------------------

   function Is_Lock_Error (The_Status : Status) return Boolean;
   function Is_Lock_Error (The_Condition : Condition) return Boolean;

   Cant_Get_Object_Lock             : constant Status := 16#0131#;
   Cant_Get_Record_Lock             : constant Status := 16#0132#;
   Cant_Get_View_Slot_Lock          : constant Status := 16#0133#;
   Cant_Get_Lock_On_Predecessor     : constant Status := 16#0134#;
```

```
   Illegal_Lock_Mode                  : constant Status := 16#0135#;
   Invalid_Object_Lock_Handle         : constant Status := 16#0136#;
   Object_Is_Not_Locked               : constant Status := 16#0137#;

   -------------------------------
   -- Load_Image errors (014x) --
   -------------------------------

   Load_Image_Is_Too_Big              : constant Status := 16#0141#;
   Not_A_Secondary_Load_Image         : constant Status := 16#0142#;

   --------------------------------
   -- Code segment errors (015x) --
   --------------------------------

   Not_A_Cross_Cg_Segment             : constant Status := 16#0151#;
   Not_In_Cross_Cg_Object_World       : constant Status := 16#0152#;
   Not_A_New_Code_Segment             : constant Status := 16#0153#;

   --------------------------------
   -- job/session errors  (016x) --
   --------------------------------

   Bad_Stack_Id : constant Status := 16#0160#;
   Bad_Job_Id : constant Status := 16#0161#;
   Bad_Session_Id : constant Status := 16#0162#;
   State_Stack_Overflow               : constant Status := 16#0163#;
   State_Stack_Underflow              : constant Status := 16#0164#;
   View_Stack_Overflow                : constant Status := 16#0165#;
   View_Stack_Underflow               : constant Status := 16#0166#;
   Invalid_Stack_Location             : constant Status := 16#0167#;

   ----------------------------------------
   -- Other identification errors (02xx) --
   ----------------------------------------

   Unit_Does_Not_Exist                : constant Status := 16#0221#;
   Parent_Object_Does_Not_Exist       : constant Status := 16#0222#;
   Predecessor_Version_Is_Not_Known   : constant Status := 16#0223#;
   Segment_Does_Not_Exist             : constant Status := 16#0224#;

   World_Already_Exists               : constant Status := 16#02a0#;
   Link_Pack_Entry_Already_Exists     : constant Status := 16#02A1#;

   -----------------------------
   -- Link Pack Errors (02Bx) --
   -----------------------------

   function Is_Link_Error (The_Status : Status) return Boolean;
   function Is_Link_Error (The_Condition : Condition) return Boolean;

   Link_Name_Does_Not_Exist       : constant Status := 16#02B0#;
   Cant_Delete_Internal_Link      : constant Status := 16#02B1#;
   Duplicate_Link_Name            : constant Status := 16#02B2#;
   External_Link_To_Local_Object  : constant Status := 16#02B3#;
   Cant_Create_Internal_Link      : constant Status := 16#02B4#;
   Cant_Create_External_Link      : constant Status := 16#02B5#;
   Cant_Delete_External_Link      : constant Status := 16#02B6#;
```

```
   Cant_Identify_Link              : constant Status := 16#02b7#;

   -------------------------
   -- View Errors (02Dx) --
   -------------------------

   function Is_View_Error (The_Status : Status) return Boolean;
   function Is_View_Error (The_Condition : Condition) return Boolean;

   Bad_Object_Index_Constraint           : constant Status := 16#02D0#;

   ------------------------------------------------
   -- Miscellaneous object operation errors (03xx) --
   ------------------------------------------------

   Illegal_Class : constant Status := 16#0301#;
   Object_Mismatch : constant Status := 16#0302#;
   Object_Class_Mismatch : constant Status := 16#0303#;
   Too_Many_Versions : constant Status := 16#0304#;
   Version_Is_Still_Referenced           : constant Status := 16#0305#;
   Multiple_Versions_Not_Supported       : constant Status := 16#0306#;
   World_View_Is_Frozen : constant Status := 16#0307#;
   Version_Is_Frozen : constant Status := 16#0308#;
   World_Still_Contains_Objects          : constant Status := 16#0309#;
   World_Still_Contains_Subworlds        : constant Status := 16#030A#;
   Object_Index_Already_In_Use           : constant Status := 16#030B#;
   Explicit_Version_Creation_Not_Allowed : constant Status := 16#030C#;
   Object_Already_Exists : constant Status := 16#030D#;
   Object_Is_Not_Reserved                : constant Status := 16#030E#;
   Must_Update_Via_View : constant Status := 16#030F#;

   ---------------------------------
   -- Resource limit errors (04xx) --
   ---------------------------------

   function Is_Resource_Limit_Error (The_Status : Status) return Boolean;
   function Is_Resource_Limit_Error (The_Condition : Condition) return Boolean;

   Object_Is_Out_Of_Version_Numbers         : constant Status := 16#0401#;

   World_Is_Out_Of_Segment_Numbers          : constant Status := 16#0410#;
   World_Is_Out_Of_Code_Segment_Numbers     : constant Status := 16#0411#;
   World_Is_Out_Of_Object_Indices           : constant Status := 16#0412#;
   World_Is_Past_Low_Space_Warning_Threshold : constant Status := 16#0413#;

   Insufficient_Object_Indices_Available    : constant Status := 16#0414#;

   Unit_Is_Past_Low_Space_Warning_Threshold  : constant Status := 16#0420#;
   Unit_Is_Out_Of_Disk_Space : constant Status := 16#0421#;

   System_Is_Out_Of_Action_Numbers          : constant Status := 16#0430#;
   System_Is_Out_Of_Job_World_Numbers       : constant Status := 16#0431#;
   System_Is_Out_Of_Object_World_Numbers    : constant Status := 16#0432#;

   System_Is_Out_Of_Session_Numbers         : constant Status := 16#0433#;


   Job_Is_At_Action_Limit : constant Status := 16#0440#;
```

```
    Job_Is_At_Action_Node_Limit                : constant Status := 16#0441#;

    ----------------------------------
    -- Access control errors (05xx) --
    ----------------------------------

    function Is_Access_Control_Error (The_Status : Status) return Boolean;
    function Is_Access_Control_Error (The_Condition : Condition) return Boolean;

    ----------------------------------
    -- Physical I/O errors (101x) --
    ----------------------------------

    Page_Does_Not_Fit_In_Cache          : constant Status := 16#1011#;
    Page_Is_Not_In_The_Cache            : constant Status := 16#1012#;
    Page_Is_Already_Wired               : constant Status := 16#1013#;
    Page_Is_Already_Loading             : constant Status := 16#1014#;
    Unrecoverable_Disk_Error            : constant Status := 16#1015#;

    --------------------------
    -- KK/OM errors (Dxxx) --
    --------------------------

    ------------------------------------------------------------
    -- Status conditions for internal KK/OM interfaces (Exxx) --
    ------------------------------------------------------------

    Page_Is_Already_Defined             : constant Status := 16#E001#;
    Page_Is_Not_Defined                 : constant Status := 16#E002#;
    Page_Is_Already_Shadowed            : constant Status := 16#E003#;
    Header_Already_Stored_In_Free_List  : constant Status := 16#E004#;

    ------------------------------------------------------
    -- Status conditions that are not even visible from --
    --   internal KK/OM interfaces (Fxxx)               --
    ------------------------------------------------------

end Error;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.ACTION
Version: V(93)
Request: 1270

Date: April 24, 1986
Queued: 11:20:12 AM
Printed: 11:30:07 AM

```
with Job;
with Sys;
with Error;
with Machine;
with Version_Map; -- private eyes
with Om_Definitions;

pragma Private_Eyes_Only;
with Action_Layout;

package Action is
    -- pragma Subsystem (Object_Management);
    -- pragma Module_Name (4, ?);

    package System renames Sys;
    package Om renames Om_Definitions;

    subtype Version_Id is Om.Version_Id;
    subtype Action_Id is Om.Action_Id;
    subtype Lock_Mode is Om.Lock_Mode;
    subtype Record_Key is Om.Record_Key;

    function Max_Length_Of_Log_Record return Natural;

    subtype Log_Record_Value is System.Bit_String;

    ------------------------------
    -- Basic action operations --
    ------------------------------

    procedure Start_Action (Id : out Action_Id;
                            Status : out Error.Condition;
                            Update_Prevented : Boolean := False;
                            Commit_Prevented : Boolean := False;
                            Guardian : Machine.Task_Id := Job.Root_Thread;
                            Master : Action_Id := Om.Nil);

    -- Used to start an action.  Update_Prevented = true is equivalent to
    -- calling Prevent_Update.  Similarly, Commit_Prevented = true is
    -- equivalent to calling Prevent_Commit.  Note that Update_Prevented
    -- makes starting and finishing the action considerably faster (since
    -- no IO is required to write commit records).

    -- The calling task is said to be the "owner" of the action.  One will
    -- observe that only the owner can operate on the action (but there is
    -- a mechanism for transferring ownership).

    -- If both the owner and the guardian die (become not 'callable), the
    -- action will eventually get auto-abandoned.  The default value for
    -- the Guardian parameter causes actions to almost never get auto abandoned
    -- (because death of the root thread implies termination of the job, and
    -- job termination implicitly invokes Abandon_All_Actions_In_Job).

    -- Specifying a Nil Master causes the new action to be a master.
    -- Specifying a non-Nil Master causes the new action to be a dependent of
    -- the named Master, and to become commit prevented.  When the master is
    -- finished (by explicit Commit, Abandon, or auto-abandon), its associated
    -- dependent is Abandon'd.  The guardian and owner of a dependent action
```

```
    -- are those of its master.

    -- Possible errors:
    --   Is_Bad_Action (Status):
    --       o Master_Action_Does_Not_Exist
    --             The identified Master does not exist.
    --       o Not_The_Action_Owner
    --             Not the owner of the Master action.
    --       o Action_Is_A_Dependent_Action
    --             The Master parameter refers to a dependent action.
    --       o Action_Already_Has_Dependent
    --             Master's can have at most one dependent.
    --   Is_Resource_Limit_Error (Status):
    --       o System_Is_Out_Of_Action_Numbers
    --       o Job_Is_At_Action_Limit
    --       o Job_Is_At_Action_Node_Limit

    -------------------------
    -- Internal operations --
    -------------------------

    procedure Acquire_Object_Lock
                    (Id : Action_Id;
                     On_Version : Version_Id;
                     Map_Location : Version_Map.Data_Item_Pointer;
                     In_Mode : Lock_Mode;
                     For_New_Version : Boolean;
                     Status : out Error.Condition;
                     Max_Wait : Duration := Duration'Last);

    type Object_Lock_Handle is private;
    Nil_Object_Lock_Handle : constant Object_Lock_Handle;

    procedure Acquire_Record_Lock_Intention
                    (Id : Action_Id;
                     On_Version : Version_Id;
                     Map_Location : Version_Map.Data_Item_Pointer;
                     In_Mode : Lock_Mode;
                     Handle : out Object_Lock_Handle;
                     Status : out Error.Condition;
                     Max_Wait : Duration := Duration'Last);

    procedure Acquire_Record_Read_Lock
                    (Id : Action_Id;
                     On_Version : Object_Lock_Handle;
                     On_Item : Record_Key;
                     Status : out Error.Condition;
                     Max_Wait : Duration := Duration'Last);

    generic
        with function Provide_Log_Entry return Log_Record_Value;

    procedure Acquire_Record_Write_Lock
          (Id : Action_Id;
           On_Version : Object_Lock_Handle;
           On_Item : Record_Key;
           Status : out Error.Condition;
           Max_Wait : Duration := Duration'Last);
```

```
-- These operations are used by an action to acquire a lock.

-- The Acquire_Object_Lock and Acquire_Record_Lock_Intention operations
-- are used to acquire object level locks.  The second operation must be
-- used prior to acquiring record locks.

-- The Acquire_Record_Read_Lock and Acquire_Record_Write_Lock operations
-- are used to acquire record locks.  The client passes in the
-- Object_Lock_Handle which was returned by a previous call to
-- Acquire_Record_Lock_Intention (which locked some object "X").  In the
-- case of Acquire_Record_Read_Lock, the call gets a Record_Read lock on
-- the record (specified by On_Item) in object "X".

-- In the case of Acquire_Record_Write_Lock, the call gets a Record_Write
-- lock on the record (specified by On_Item) on object "X".  In addition,
-- once the Record_Write lock has been acquired the client must supply
-- the "log record", via Provide_Log_Entry, which will be stored in the
-- action log, and processed in order to commit/abandon the action.
-- The Provide_Log_Entry operation should not acquire any action locks
-- or mutex's, else you risk getting the action manager deadlocked.

-- The appropriate calls to the Heap_Segment_Manager are made.  In
-- particular, For Acquire_Object_Lock: if For_New_Version is true, then
-- calls Create_New_Version; else calls Open_For_Read or
-- Spawn_New_Generation, as appropriate.  Similarly, for
-- Acquire_Record_Lock_Intention, calls Open_For_Read or
-- Spawn_New_Generation, as appropriate:  Assumes caller does NOT
-- currently hold the mutex in the supplied version map entry.  Will
-- acquire and release the mutex once the object has been locked.
-- Modifies fields of the segment descriptor and version map as specified
-- by the Heap_Segment_Manager.

-- Lock compatibility matrix:
--
--                                  Current Lock
--                                (by different action)
--                        RO   WO   RRO   RWO   RR   RW   SRO   U
--                      ---------------------------------------
--            RO      :  X         X                        X    X
--                    :
--            WO      :                                    1    X
--                    :
-- Desired    RRO     :  X         X     X                  X    X
--  Lock              :
--            RWO     :            X     X                  1    X
--                    :
--            RR      :                        X          X    X
--                    :
--            RW      :                                   1    X
--                    :
--            SRO     :  X    1    X     1     X    1     X    X
--                    :
--            U       :  X    X    X     X     X    X     X    X

-- Absence of an "X" indicates that the desired access will not be granted
-- if any OTHER action (not including requesting action) has the indicated
-- current access.  The 1 indicates that while this package does not
```

```
-- allow compatibility, the Object package will "do the right thing" to
-- allow users the view that they are compatible.  Via Max_wait, queueing
-- is available when access is denied.

-- Lock upgrade matrix:
--
--                                  Current Lock
--                                 (by same action)
--                        RO    *O    RRO   RWO   RF    RW    SRO   U
--                      ------------------------------------------------
--            RO      |  X     *     X                             X     X
--                    |
--            WO      |  *2    *                                   X     X
--                    |
-- Desired    RRO     |  X           X     X                       X     X
--  Lock              |
--            RWO     |              X     X                       X     X
--                    |
--            RF      |                          X     *     X     X
--                    |
--            RW      |                          *2    *     X     X
--                    |
--            SRO     |  X     X     X     X     X     X     X     X
--                    |
--            U       |  X     X     X     X     X     X     X     X

-- Absence of an "X", "*", or "*2" indicates that the desired access
-- will not be granted.  Presence of one of these indicators indicates
-- that the desired access will be granted, even though there are other
-- actions waiting for incompatable access; but note that an action can
-- upgrade from RO to *O iff it is the only action with RO.  The three
-- positions with "*"s identify the actual upgrade cases.  Note that there
-- are "X"s in the same positions as in the compatibility matrix.

-- The semantics of Supersedeable_Read_Object are as follows: Obviously,
-- if the version in question does not undergo updates while the s-reader
-- has the version, the s-reader sees the same value as a vanilla reader.
-- The remaining cases: (1) The version is open for update when the
-- s-reader arrives. In this case, reader sees the writer's before image,
-- and continues to see it even after the writer commits. The
-- implementation will cause the reader to make a full copy of the before
-- image at open time. (2) The version is already open by an s-reader when
-- the write arrives. The reader continues to see the same image. The
-- writer simply makes a new version, in a fashion similar to the implicit
-- versions caused by various version control policies. Implementation
-- note: most of the implementation of supersedeable read actually happens
-- in the Object package.

-- The semantics of Unsychronized are as follows: If the version is not
-- already open, the u-opener will have no access. If the version is
-- already open, the u-opener will have access to the same generation as
-- the other opener's.  Take note that this means that the u-opener has
-- access to the same generation as a writer, and can in fact make changes
-- to the writer's image.  When the version is closed by all of the other
-- opener's, the u-opener will have no access. The u-opener should be
-- capable of dealing with Non_Existent_Page_Error at any time, since
-- access may go away at any time. The above semantics may change from
-- software release to software release, so don't write lots of code that
```

```
    -- depends on it. In particular, this mode should NOT be directly
    -- available to customers (exception: ok when shielded by the IO
    -- packages).

    -- In some cases, the action manager considers the action to be in
    -- deadlock, or near deadlock, and returns Lock_Error immediately, even
    -- thought the acquire operation gave a non-zero Max_Wait. Example:
    -- Actions A1 and A2 hold RO locks on object X; action A2 attempts to
    -- upgrade to WO, but can't because of the RO held by A1; action A2
    -- will get Lock_Error immediately. Another example: Another example:
    -- Action A holds a WO lock on object X and attempts to get a RWO lock
    -- on X with non-zero Max_Wait; it will get Lock_Error immediately.

    -- ***WARNING*** The presence of the upgrades marked by "*" allows for
    -- strange behaviour.  2 scenarios are described below:

    -- Consider: task T1 uses action A to get a record write lock on some
    -- record X; task T1 passes ownership to task T2 which proceeds to also
    -- get a record write lock on the same record X; tasks T1 and T2 each
    -- proceed to operate on the record concurrently; unless these tasks are
    -- prepared for this, one may get strange results. Tasks can protect
    -- themselves via the following technique: acquire the lock (which may be
    -- via upgrade) and do not relinquish ownership of the action until after
    -- leaving the "critical region".

    -- Consider a example which involves just a single task: Procedure X has a
    -- handle for action A, gets a "file handle" for write on object O, does
    -- work (like building up context in the file), and calls some procedure
    -- Y, passing it A.  The exact effect of procedure Y is not precisely
    -- known by X.  Y gets yet another "file handle" for write on object O
    -- (this is allowed by the upgrade semantics), and completely changes the
    -- contents of object O.  On return, X gets very confused. The only known
    -- solution is for applications to understand whats going on.

    -- These operations are only available to the OM subsystem, and to other
    -- "registered" environment clients (like the IO packages). In particular,
    -- these operations are not available to "users".

    -- Possible errors:
    --   Is_Bad_Action (Status):
    --       o Action_Does_Not_Exist
    --       o Not_The_Action_Owner
    --       o Update_Has_Been_Prevented
    --   Is_Lock_Error (Status):
    --       o Cant_Get_Object_Lock (1st & 2nd ops only)
    --       o Cant_Get_Record_Lock (3rd and 4th ops only)
    --       o Illegal_Lock_Mode
    --            The Acquire_Object_Lock operation only takes modes
    --            Read_Object and Write_Object.  The
    --            Acquire_Record_Lock_Intention operation only takes modes
    --            Record_Read_Object and Record_Write_Operation.
    --   Is_Resource_Limit_Error (Status):
    --       o Job_Is_At_Action_Node_Limit

    procedure Acquire_Record_Read_Lock_And_Object_Lock
                  (Id : Action_Id;
                   Record_Read_Lock_Version : Object_Lock_Handle;
                   On_Item : Record_Key;
```

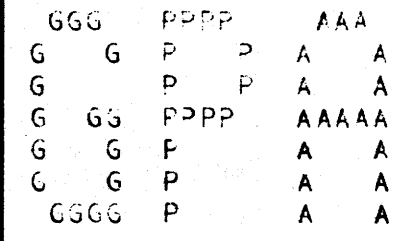
```
                    Object_Lock_Version : Version_Id;
                    Map_Location : Version_Map.Data_Item_Pointer;
                    Object_Mode : Lock_Mode;
                    Status : out Error.Condition;
                    Max_wait : Duration := Duration'Last);

    procedure Acquire_Record_Read_Lock_And_Intention_Lock
                   (Id : Action_Id;
                    Record_Read_Lock_Version : Object_Lock_Handle;
                    On_Item : Record_Key;
                    Object_Lock_Version : Version_Id;
                    Map_Location : Version_Map.Data_Item_Pointer;
                    Object_Mode : Lock_Mode;
                    Handle : out Object_Lock_Handle;
                    Status : out Error.Condition;
                    Max_Wait : Duration := Duration'Last);

    -- These operations are a composite of Acquire_Record_Read_Lock and
    -- Acquire_Object_Lock or Acquire_Record_Read_Lock and
    -- Acquire_Record_Lock_Intention.  Either both locks are acquired or
    -- neither is.

    procedure Append_Log_Entry (Id : Action_Id;
                                On_Version : Version_Id;
                                On_Item : Record_Key;
                                Log_Record : Log_Record_Value;
                                Status : out Error.Condition);

    -- In some special cases, additional information is needed to abandon an
    -- action.  If On_Item is nil, the action must have a Write_Object lock
    -- on the version.  If On_Item is not nil, the action must have a
    -- Record_Write_Object lock on the version and a Record_Write lock on
    -- On_Item.  Currently, this operation is only used by load image's.

    procedure Re_Open_Segment (On_Version : Version_Id;
                               Map_Location : Version_Map.Data_Item_Pointer;
                               Status : out Error.Condition);

    -- Due to the internal workings of commit in the face of multiple writers
    -- on the same object, tasks will occasionally handle
    -- Write_To_Read_Only_Page conditions by calling this routine.

    -- This operation is available only to DM subsystems.

    -- Possible errors:
    --   Version_Is_Frozen

    procedure Abandon_All_Actions_In_Job (The_Job : Machine.Job_Worlds;
                                          Status : out Error.Condition);

    -- Used by Job termination to abandon every action created and still open
    -- by the specified job. Assumes the job's modules have already been
    -- terminated.

    -- This operation is available only to DM subsystems.

    -- Possible errors:
    --   Is_Bad_Actior (Status);
```

```
--       o Action_Owner_Not_Dead
--             o Will get this if there exists a task (in The_Job, other
--               than the thread running this procedure) which still
--               claims to be owner of an action owned by The_Job.
--       o Action_Guardian_Not_Dead
--             o Will get this is the guardian task is not dead.

-----------------------------
-- Basic action operations --
-----------------------------

procedure Commit (Id : Action_Id;
                  Status : out Error.Condition);

-- None of the updates of this action actually take place until Commit
-- is called.

-- Possible errors:
--   Is_Bad_Action (Status):
--       o Action_Does_Not_Exist
--       o Not_The_Action_Owner
--       o Commit_Has_Been_Prevented
--       o Action_Is_A_Dependent_Action

procedure Abandon (Id : Action_Id;
                   Status : out Error.Condition);

-- Used to undo the updates of this action.  Implicitly called when the
-- owning task/job is abnormally terminated, and when the system crashes.

-- Possible errors:
--   Is_Bad_Action (Status):
--       o Action_Does_Not_Exist
--       o Not_The_Action_Owner
--       o Action_Is_A_Dependent_Action

procedure Prevent_Commit (Id : Action_Id;
                          Status : out Error.Condition);

-- Used to prevent an action from being committed.  Useful when an
-- error is detected at the "bottom" of an algorithm, and it wants to
-- guarantee that erroneous changes to open objects are not committed.
-- Ok if the action is already commit-prevented.

-- Possible errors:
--   Is_Bad_Action (Status):
--       o Action_Does_Not_Exist
--       o Not_The_Action_Owner

procedure Prevent_Update (Id : Action_Id;
                          Status : out Error.Condition);

-- Used to prevent an action from acquiring an Object_Write or Record_Write
-- lock on any object.  This is implicitly called by Object.Open when the
-- Supersedable parameter is true.  Ok if the action is already
-- update-prevented.

-- Possible errors:
```

```
--    Is_Bad_Action (Status):
--        o Action_Does_Not_Exist
--        o Not_The_Action_Owner
--        o Action_Already_Has_Update_Locks

procedure Give_Up_Ownership (Id : Action_Id;
                             Status : out Error.Condition);

procedure Take_Ownership (Id : Action_Id;
                          Status : out Error.Condition);

-- These operations are used to cause ownership to be transfered from one
-- task to another.  The current owner (task T1) executes Give_.  Any
-- future operations by T1 will result in Not_The_Action_Owner. The
-- proposed new owner (task T2) then executes Take_ and becomes the new
-- owner.

-- In other words, once Give_ has been called, the only exported
-- operation that can be successfully applied to the action is Take_.  In
-- the window between the call to Give_ and the call to Take_, the
-- action is protected (from auto-abandonment) by the continued existance
-- of task T1. The new owner must be in the same job as the previous owner.

-- If a task is currently the owner, and dies, it does an implicit Give_.

-- Possible errors:
--    Is_Bad_Action (Status):
--        o Action_Does_Not_Exist
--        o Not_The_Action_Owner (1st op only)
--        o Action_Owner_Not_Willing (2nd op only)
--        o Cant_Xfer_Ownership_Cross_Jobs (2nd op only)
--        o Action_Is_A_Dependent_Action

--------------------------
-- Information requests --
--------------------------

function Is_In_Progress (Id : Action_Id) return Boolean;

-- When false, the reason could be one of: (a) The given Id is
-- garbage, and has never been started, (b) the action has committed,
-- and (c) the action has abandoned.  There are no exported ways to
-- distinguish between these cases.

function Is_Commit_Prevented (Id : Action_Id) return Boolean;
function Is_Update_Prevented (Id : Action_Id) return Boolean;

-- Returns the corresponding attributes of the action.  Returns false for
-- garbage input.

function Is_Locked (The_Version : Version_Id) return Boolean;
function Is_Locked (The_Version : Version_Id;
                    The_Item : Record_Key) return Boolean;

-- True iff the object/item is currently locked (in some mode, other
-- than Unsynchronized).  Returns false for garbage Version_Id.

function Owner (Id : Action_Id) return Machine.Task_Id;
```

```
    function Owner (Id : Action_Id) return Om.Job_Id;

    -- Returns Nil when the given action is not in progress, or the Action_Id
    -- is garbage.

    function Guardian (Id : Action_Id) return Machine.Task_Id;

    -- Returns Nil when the given action is not in progress, or the Action_Id
    -- is garbage.

    function Master (Id : Action_Id) return Action_Id;

    -- Returns Nil when the given action is a master, is not in progress, or
    -- the Action_Id is garbage.

    function Dependent (Id : Action_Id) return Action_Id;

    -- Returns Nil when the given action is not a master, is not in progress,
    -- or the Action_Id is garbage.

    function Consumed_Log_Block (Id : Action_Id) return Integer;
    function Action_Nodes (Id : Action_Id) return Integer;

    -- Returns 0 when the given action is not in progress, or the Action_Id
    -- is garbage.  These values reflect how much of the Action_Manager's
    -- resources are being consumed by the given action.

    generic
        with procedure Visit (Locker_Id : Action_Id;
                              Locker_Job : Om.Job_Id;
                              In_Mode : Lock_Mode);

    procedure Traverse_Lockers (The_Version : Version_Id;
                                The_Item : Record_Key);

    -- Used to find all the actions which have some lock on the given
    -- version/item.  Calls Visit iff the version/item is locked in some
    -- mode other than Unsynchronized.  May call Visit more than once for the
    -- same action.  Noop for garbage input.

    generic
        with procedure Visit (The_Version : Version_Id;
                              The_Item : Record_Key;
                              In_Mode : Lock_Mode);

    procedure Traverse_Locks (Id : Action_Id);

    -- Used to find all the locks currently held by the given action. Does not
    -- call Visit if the given action is not in progress. In_Mode will never
    -- be Unsynchronized.  May call Visit more than once for the same
    -- version/item. Noop for garbage input.

    procedure Set_Daemon_Wait_Interval (Interval : Duration);

private
    type Object_Lock_Handle is new Action_Layout.Lock_List_Pointer;
    Nil_Object_Lock_Handle : constant Object_Lock_Handle := null;
```

```
end Action;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.NAMING
Version: V(41)
Request: 1271

Date: April 24, 1986
Queued: 11:26:18 AM
Printed: 11:36:50 AM

```
with Job;
with Error;
with Om_Definitions;

package Naming is
    -- pragma Subsystem (Object_Management);

    -- Note that there is a parallel package, Naming_Messages, which will
    -- interpret bad status and produce error message strings.

    package Om renames Om_Definitions;

    subtype Context is Om.Naming_Context;

    ---------------------
    -- Name resolution --
    ---------------------

    -- The rules from TRW's Gamma Max document need to be reproduced here.

    -- Additional feature: Assume that we have a name of the form
    -- "...Foo'V(...). Let "!...Kernel" denote the root of the world
    -- containing the object identified by "...Foo". The following are
    -- examples of uses of the "'V" attribute:
    --
    --   (a) ...Foo'V(7)                  refers to version 7 of Foo
    --
    --   (b) ...Foo'V(Gamma.Green)        "!...Kernel.Gamma" is a directory
    --                                    with subclass Path;
    --                                    "!...Kernel.Gamma.Green" is a
    --                                    universe view; assuming it refers
    --                                    to a world view which refers to version
    --                                    14 of Foo, then the given string refers
    --                                    to version 14 of Foo.
    --
    --   (c) ...Foo'V(Gamma)              same as "...Foo'V(Gamma.Current)"
    --
    --   (d) ...Foo'V(_View'V(3))         "!...Kernel._View'V(3)" is a world
    --                                    view; assuming it refers to version 21
    --                                    of Foo, then the given string refers to
    --                                    version 21 of Foo.
    --
    -- Note that "...Foo'V(Gamma)" is not just a shorthand for "...Foo'V(7)".
    -- In particular, as the destination of an operation (say "Copy", for
    -- example), "...Foo'V(Gamma)" can cause the new version to be created in
    -- the Gamma view.

    -- We do not currently support full pathnames or complex wildcards in the
    -- "'V" attribute. In particular, the only special characters supported
    -- in the "'V" attribute are "_" and "#".

    -- Additional feature: Rational reserves all simple names which both start
    -- and end with an underscore. The empty string is not considered well
    -- formed.

    procedure Resolve (Name : Om.Unique_Wildcard;
                       Action : Om.Action_Id;
                       Result : out Om.Version_Handle;
```

```
                    Status : out Error.Condition;
                    In_Context : Context := Job.Default_Context;
                    Max_Wait : Duration := Job.Default_wait);

   -- To be successfully resolved, the given wildcard must match exactly
   -- one object. Exception: search lists cause success on first match.

   -- The object referenced by the given wildcard is identified by
   -- Result.Version. If the given wildcard specifies a specific version
   -- number (by using "'V(7)", for example), then Result.Version.Number
   -- identifies that particular version; otherwise, Result.Version.Number is
   -- Nil. If the given wildcard specifies a universe/world view (by using
   -- "'V(Gamma)", for example), then Result.Universe identifies that
   -- particular universe/world view; otherwise, Result.Universe is Nil.
   -- Note that the syntax is such that Result.Version.Number and
   -- Result.Universe cannot both be non-Nil.  These rules are such that
   -- operations like Object.Open "do the right thing".

   -- Action and Max_Wait are used iff Name refers to indirect files.

   -- Note that this form of resolve does NOT produce Diana.Tree.

   -- Possible errors:
   --   Is_Bad_Object_Id (Status)
   --       o These are caused by a context which does not identify an
   --         existing object.  These are possible iff Name does not
   --         start with "!".
   --   Is_Bad_Pathname (Status)
   --   Is_Bad_Action (Status)


   generic
       with procedure Visit_Roof (Roof : Om.Object_Handle);
       with procedure Visit (Result : Om.Version_Handle);
       with procedure Handle_Error (Status : Error.Condition);


   procedure Wild_Resolve
        (Name : Om.Full_Wildcard;
         Action : Om.Action_Id;
         In_Context : Context := Job.Default_Context;
         Suppress_Duplicates : Boolean := False;
         Max_Versions : Integer := 2 ** 16;
         Calculate_Roof : Boolean := False;
         Max_wait : Duration := Job.Default_Wait);

   -- Similar to the Resolve procedure, with these differences:  Is happy
   -- with 0 or more matches, calling Visit for each match.  After an error,
   -- this procedure will simply stop searching further in the active context,
   -- going back to any remaining contexts in its stack.  The caller is free
   -- to raise arbitrary exceptions in the generic formal subprograms; these
   -- terminate the resolution.

   -- When Suppress_Duplicates is false, it is possible for sets, indirect
   -- files, and wildcarded universes to cause the same object to be Visit'd
   -- more than once.  When true, this procedure has to build up an internal
   -- map to filter out duplicates; this has considerable performance impact.
```

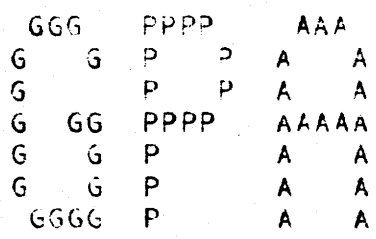
```
    -- The Visit_Roof procedure is called iff Calculate_Roof is true.  In this
    -- case, Visit_Roof is called prior to any calls to Visit.  A "roof" is
    -- an object which is a parent of every object identified by Visit.  This
    -- routine attempts to find the "best roof", namely the parent which
    -- if furthest from the root.  Use of this feature causes the procedure
    -- to build up an internal map and has considerable performance impact.

    -- Max_Versions is used if either Suppress_Duplicates or Visit_Roof is
    -- true.  It limits the size of the internal map and therefore the number
    -- of versions which can result from the resolution.  If the error
    -- Too_Many_Versions is returned, resolution stops, and Visit is called
    -- for all of the versions currently in hand.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --       o Something wrong with In_Context.
    --   Is_Bad_Pathname (Status)
    --       o Something wrong with Name.
    --   Too_Many_Versions
    --       o Max_Versions is smaller than the number of objects which
    --         match the wildcard.  Cannot be returned unless
    --         Suppress_Duplicates or Calculate_Roof is true.
    --   Is_Bad_Action (Status)


    type Create_Kind is new Long_Integer range 0..2 ** 3 - 1;
    Creating_New_Version : constant Create_Kind := 1;
    Creating_New_Object  : constant Create_Kind := 2;
    Creating_Temp_Object : constant Create_Kind := 3;

    procedure Resolve_For_Create (Name : Om.Unique_Wildcard;
                                  Action : Om.Action_Id;
                                  Kind : out Create_Kind;
                                  Result : out Om.Object_Handle;
                                  Child_Name : out Om.Simple_Name;
                                  Class_Of_New_Object : out Om.Class_Number;
                                  Status : out Error.Condition;
                                  In_Context : Context := Job.Default_Context;
                                  Max_Wait : Duration := Job.Default_Wait);

    -- This operation is used by the string form of Object.Create. Assuming
    -- Status is is Successful: If the given Name resolves to an already
    -- existing object, then Kind will be Creating_New_Version, and Result
    -- will identify the object (for which a new version is to be created),
    -- just as if the vanilla Resolve had been invoked. If the string is empty
    -- (with the possible exception of a "'C" attribute) then Kind will be
    -- Creating_Temp_Object, Child_Name will be a simple name string generated
    -- from a unique id (returned by Om.Get), and Class_Of_New_Object will be
    -- Nil (or the value of the optional "'C" attribute). Otherwise, Kind will
    -- be Creating_New_Object, Result will identify the parent of the new
    -- object, Child_Name will contain the simple name of the new object, and
    -- Class_Of_New_Object will be Nil (or the value of the optional "'C"
    -- attribute).

    -- Possible errors:
    --   includes all those that can be returned by vanilla Resolve
```

```
   procedure Substitute (The_Object : Om.Version_Handle;
                         Source : Om.Full_Wildcard;
                         Destination : Om.Full_Wildcard;
                         Result : out Om.Pathname;
                         Status : out Error.Condition);

   -- Given a source name (with wild cards) that matches the name of the
   -- given object, return a Result string in which substitution characters
   -- have been replaced by the matching portions of the object's name as
   -- indicated by the destination pattern.

   -- Substitution semantics need to be fully specified.

   -- Used by Object.Wild_Copy.

   -- RESTRICTION: not supported in 1st qtr impl.

   -- Possible errors:
   --   Is_Bad_Version_Handle (Status)
   --       o Something wrong with The_Object.
   --   Is_Bad_Pathname (Status)
   --       o Something wrong with Source or Destination.  Use the
   --         Is_Well_Formed function to see which one is bad.

   ---------------------------------
   -- Detailed structure of names --
   ---------------------------------

   function Is_Well_Formed (Name : Om.Full_Wildcard) return Boolean;
   procedure Check_Well_Formed (Name : Om.Full_Wildcard;
                                Status : out Error.Condition);

   -- Possible errors:
   --   Filename_Syntax_Error
   --       o Status identifies the character in Name which caused the error,
   --           as well as the token class which was expected.

   function Prefix (Name : Om.Full_Wildcard) return Om.Full_Wildcard;
   function Head (Name : Om.Full_Wildcard) return Om.Full_Wildcard;
   function Tail (Name : Om.Full_Wildcard) return Om.Full_Wildcard;

   -- Given ill formed names, these functions produce undefined results.
   -- Prefix removes the last segment.  Head returns only the first segment.
   -- And Tail removes the first segment.
   --    Prefix ("A.B.C") => "A.B"
   --    Prefix ("A") => ""
   --    Head ("A.B.C") => "A"
   --    Head ("A") => "A"
   --    Tail ("A.B.C") => "B.C"
   --    Tail ("A") => ""

   function Attributes (Name : Om.Full_Wildcard) return String;
   function Class_Attribute (Name : Om.Full_Wildcard) return String;
   function Version_Attribute (Name : Om.Full_Wildcard) return String;
   function Part_Attribute (Name : Om.Full_Wildcard) return String;

   -- Given ill formed names, these functions produce undefined results.
   -- Attributes returns the attributes at the end of the Name, including the
```

```
    -- leading "'", or a null string if Name has no a attributes.  The
    -- remaining functions return the indicated attribute.

    procedure Get_Full_Name (The_Object : Om.Object_Handle;
                             Full_Name : out Om.Pathname;
                             Status : out Error.Condition);

    procedure Get_Simple_Name (The_Object : Om.Object_Handle;
                               Name : out Om.Simple_Name;
                               Status : out Error.Condition);

    -- Returns the full/simple name for The_Object, excluding qualifying
    -- attributes (like the "'V" attribute). The Version_Number component of
    -- The_Object is ignored.

    -- Possible errors:
    --    Is_Bad_Version_Handle (Status)

    procedure Get_Unique_Full_Name (The_Object : Om.Version_Handle;
                                    Full_Name : out Om.Pathname;
                                    Status : out Error.Condition);

    procedure Get_Unique_Simple_Name (The_Object : Om.Version_Handle;
                                      Name : out Om.Simple_Name;
                                      Status : out Error.Condition);

    -- Returns the fully/simple name for The_Object, including qualifying
    -- attributes (like the "'V" attribute).

    -- Possible errors:
    --    Is_Bad_Version_Handle (Status)

    ------------------------------------
    -- Detailed directory structure --
    ------------------------------------

    function Root return Om.Object_Handle;

    -- Returns the Object_Id for "!".

    procedure Get_Parent (Child : Om.Object_Handle;
                          Parent : out Om.Object_Handle;
                          Status : out Error.Condition);

    -- Returns the handle of the Child's parent.  If Child refers to "!", then
    -- returns Nil_Object_Handle.

    -- Possible errors:
    --   Is_Bad_Object_Id (Status)

    procedure Get_Containing_Directory
                   (Of_Object : Om.Object_Handle;
                    Containing_Directory : out Om.Object_Handle;
                    Status : out Error.Condition);
    procedure Get_Containing_World
                   (Of_Object : Om.Object_Handle;
                    Containing_World : out Om.Object_Handle;
                    Status : out Error.Condition);
```

```
    procedure Get_Associated_Directory
                  (For_Object : Om.Object_Handle;
                   Associated_Directory : out Om.Object_Handle;
                   Status : out Error.Condition);
    procedure Get_Associated_World
                  (For_Object : Om.Object_Handle;
                   Associated_World : out Om.Object_Handle;
                   Status : out Error.Condition);

    -- The Containing_ functions return the nearest enclosing world/directory.
    -- The Associated_ functions are the same as the Containing_ functions,
    -- except if the argument is a world/directory, the argument is returned.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)

    procedure Get_Child (Parent : Om.Object_Handle;
                         Child_Name : Om.Simple_Wildcard;
                         Child_Id : out Om.Object_Handle;
                         Status : out Error.Condition);

    -- Returns the Object_Handle of the one child (of parent) whose simple
    -- name matcnes Child_Name (which may include wildcards).

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --      o Something wrong with Parent
    --   Is_Bad_Pathname (Status)
    --      o Something wrong with Child_Name

    generic
        with procedure Visit (Child_Id : Om.Object_Handle;
                              Child_Name : Om.Simple_Name);
    procedure Traverse_Children (Parent : Om.Object_Handle;
                                 Status : out Error.Condition);

    -- Calls Visit for every child object of Parent.  Visit is called in
    -- sort order, by Child_Name.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --      o Something wrong with Parent


    generic
        with procedure Visit (Depth : Positive;
                              The_Object : Om.Version_Handle;
                              Its_Name : Om.Simple_Name_String;
                              Go_Deeper : out Boolean);
        with procedure Handle_Error (Status : Error.Condition);

    procedure Depth_First_Traversal
         (Root_Of_Traversal : Om.Object_Handle;
          Pattern : Om.Simple_Wildcard := "@";
          Class : Om.Class_Number := Om.Nil;
          Subunits : Boolean := True;
          Directories : Boolean := True;
          Worlds : Boolean := True;
```

```
        Versions : Boolean := False);

    -- Calls Visit for Root_Of_Traversal, with Depth = 1.  Its children will
    -- have a depth of 2, etc.  Child objects are visited in sort order by
    -- Its_Name.  Versions are visited in sort order by Version_Number. Assume
    -- that objects "C1" and "C2" are children of object "P". "depth first"
    -- implies that all of C1's children (including all their  versions) are
    -- Visit'd prior to visiting C2 or its children.  It also implies that all
    -- versions of C1 are visited prior to any version of C2.

    -- The Go_Deeper out parameter of Visit can be used to prune the traversal.

    -- The Pattern, Class, Subunits, Directories, Worlds and Versions
    -- parameters are filters which prune the traversal.

    -- The caller is free to raise arbitrary exceptions in the generic formal
    -- subprograms; these terminate the resolution.

    -- Possible errors:
    --   Is_Bad_Version_handle (Status)
    --      o Something wrong with Root_Of_Traversal
    --   Is_Bad_Pathname (Status)
    --      o Something wrong with Pattern

end Naming;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.OBJECT
Version: V(173)
Request: 1272

Date: April 24, 1986
Queued: 11:26:30 AM
Printed: 11:37:20 AM

```
with Job;
with Error;
with Action;
with Machine;
with Calendar;
with Om_Definitions;

package Object is
    -- pragma Subsystem (Object_Management);

    package Om renames Om_Definitions;

    subtype Unique_Id is Om_Definitions.Unique_Id;

    subtype Context is Om.Naming_Context;

    ----------------------
    -- Basic Operations --
    ----------------------

    -- Normally, objects have 1 or more versions; that is, expunging the last
    -- version of an object will normally cause the object itself to be
    -- expunged. However, the user model includes the ability for objects to
    -- exist with no versions. In particular, when the last version of an
    -- object is expunged, but the object still has children, then the parent
    -- object will continue to exist, with no versions. When the child objects
    -- are expunged, the parent object (with no versions) will also be
    -- expunged).

    -- A view can select a version of a child object without selecting a
    -- version of the parent object.

    -- Note that the above semantics apply to directories as well as other
    -- objects (since all object classes may have children).

    -- The phrase "usual rules for Naming.Resolve" means that the operation
    -- takes 3 parameters:
    --          Name : Om.Unique_Wildcard;
    --          Action : Om.Action_Id;
    --          In_Context : Context := job.default_context;
    -- These parameters are passed to Naming.Resolve to compute the
    -- Version_Handle. Typically a second form of the operation is also
    -- provided which takes a Version_Handle, to avoid calling Naming.Resolve.

    -- The term "retention list" refers to the set of versions selected by
    -- a world view.  The "current" version (selected by a world view) is
    -- the "top" item of the retention list.  "Pushing" causes a new version
    -- to be added to the top of the retention list, becoming current;
    -- note however that if current is Nil, pushing simply causes the new
    -- version to replace current. If the retention list was already full (as
    -- specified by the retention count in force for the world view), then
    -- pushing may also cause the "bottom" item to be removed from the list,
    -- the removed version is expunged if possible (as defined by the Expunge
    -- procedure later in this spec). The retention list (together with the
    -- Reserved bit) is also referred to as the "slot" for the object.

    -- The phrase "usual rules for interpretation of Version_Handle" means:
    --   (a) As the source of an operation (or the target of an operation
```

```
--          which does not create a new version): If
--          Version_Handle.Version.Number is not Nil, then the specified
--          version is used, and Version_Handle.Universe is ignored.
--          Otherwise, we use the Version_Number which is current in the
--          selected world view.  The "selected view" is specified by the
--          .Universe component or Job.Default_world_View when the .Universe
--          component is Nil. A Record_Read lock is acquired on the slot.
--      (b) As the target of an operation which creates a new version,
--          Version_Handle.Version.Number is ignored, and the new version is
--          created in the selected universe/world view; the definition of
--          "selected" is identical to that above. A Record_Write lock is
--          acquired on the slot. (If the Record_Write lock fails because
--          the action already has Object_Write, updates to world views
--          will use the Object_Write lock instead.)

--  The phrase "usual rules for creation of new version" means:

--  "Explicit version creation" (calling Create) causes the new
--  version to be pushed onto the retention list (for the object in the
--  selected world view), thereby becoming its "current version". This may
--  cause the bottom predecessor version to become no longer selected by
--  the world view (as required by the retention count); this predecessor
--  version will be expunged at action commit time. A Record_Write lock is
--  acquired on the slot.

--  "Implicit version creation" (calling Open with an update mode and
--  appropriate Version_Control_Policy) is identical to the above case when
--  the Version_Control_Policy does NOT have the
--  Implicit_Versions_Replace_Current bit set.  Otherwise, the new version
--  simply replaces the current version in the retention list; the replaced
--  version is expunged at commit time; and a Record_Write lock is acquired.

--  Included in the usual rules for new versions are the following Ada
--  specific properties: If the object has Class_Ada and a parent which is
--  not Class_Ada, and the "current version" (in the selected world view)
--  was previously Nil, then an appropriate Link_Pack entry is made. If the
--  selected view's Link_Pack is currently shared with other views, then a
--  new version is made (this copy is committed immediately, and will
--  therefore not be undone by action abandon). This latter rule is
--  referred to as the "usual rule for undifferentiating the Link_Pack".

--  Unless specified otherwise, all locks are acquired on behalf of the
--  given Action, and the operation will suspend for up to Max_Wait for
--  each lock. The order in which operations acquire locks is not specified.

--  Unless specified otherwise, one can assume that an operation which
--  fails leaves "the world" the way it was prior to the operation.
--  An exception: when an operation fails, it does NOT release locks that
--  it acquired prior to the failure; if no modifications have been made,
--  the action simply holds extraneous locks; if modifications have been
--  made, the action will become Commit_Prevented.


procedure Resolve (The_Object : Om.Version_Handle;
                   The_Version : out Om.Version_Id;
                   Status : out Error.Condition);

--  Takes the given Version_Handle and computes the actual Version_Id
```

```
    -- according to the usual rules for interpretation of source
    -- Version_Handle's.

    -- Possible_Errors:
    --   Is_Bad_Version_Handle (Status);


    type Open_Handle is private;
    Nil_Open_Handle : constant Open_Handle;

    procedure Create (Name : Om.Unique_Wildcard;
                      Action : Om.Action_Id;
                      Handle : out Open_Handle;
                      Status : out Error.Condition;
                      Class : Om.Class_Number := Om.Nil;
                      In_Context : Context := Job.Default_Context;
                      Initialize : Boolean := False;
                      Subclass : Om.Subclass_Number := Om.Nil;
                      Version_Policy : Om.Version_Control_Policy := Om.Nil;
                      Index : Om.Object_Index := Om.Nil;
                      Creation_Id : Om.Net_Unique_Id := Om.Nil_Net_Unique_Id;
                      Max_Wait : Duration := Job.Default_Wait);

    procedure Create (The_Object : Om.Object_Handle;
                      Action : Om.Action_Id;
                      Handle : out Open_Handle;
                      Status : out Error.Condition;
                      Initialize : Boolean := False;
                      Subclass : Om.Subclass_Number := Om.Nil;
                      Max_Wait : Duration := Job.Default_Wait);

    -- In the first form, Naming.Resolve is used to compute the Object_Handle.
    -- If this succeeds, then a new version of the object is created.

    -- Otherwise, if the Name is null (except for an optional "'C" attribute),
    -- then a "temporary object" is created. In this case, In_Context is
    -- ignored. The object is created in the Job.Garbage_Directory with a
    -- unique name constructed from the time and date. The given action
    -- becomes commit prevented.

    -- Otherwise, Prefix(Name) and In_Context are used to compute the
    -- Object_Id of the parent (of the object to be created); the last segment
    -- is used as the simple name of the child object (to be created).

    -- For a newly created object, the "'C" attribute may be used to specify
    -- the class of the object, as well as the Class parameter.

    -- For a newly created object, the Version_Policy parameter specifies
    -- how future operations on the object will behave. All versions of the
    -- object will follow the same policy.  If a Nil Version_Policy is supplied
    -- the Default_Version_Policy is used.

    -- For a newly created object, the Reserved bit is set.

    -- For a newly created object, it is possible to specify the desired
    -- Object_Index and Creation_Id. This facility is used to restore an
    -- an object from "archive" or move it from machine to machine, in the
    -- context of a particular view or world.
```

```
-- In the second form, The_Object refers to an existing object, for
-- which a new version is created.

-- The following rules apply to both forms:

-- Usual rules for interpretation of the Object_Handle.  Usual rules for
-- creation of a new version.

-- The Version function can be used (with the the returned Open_Handle) to
-- determine the id of the newly created version. The Version_Number field
-- (of the id) will be non-Nil.  The new version is held with an
-- Object_Write lock, and is open.

-- If Initialize is false, the new version is empty, and we say that "an
-- empty version was created". If the operation creates the first version
-- of a new object, the Initialize parameter is treated as if it were
-- false. Otherwise, an Object_Read lock is acquired on the previous
-- version (selected by the selected world view) and its data bits are
-- copied into the newly created version, and we say that "a non-empty
-- version was created".

-- The Predecessor_Update_Id function can be used (on the returned
-- Open_Handle) to determine the stamp of the copied version.

-- If a non-Nil value is supplied for Subclass, then the subclass of the
-- new version is set as specified (note that, unlike class, different
-- versions may have different subclasses). If a Nil value is supplied,
-- and Initialize is treated as True, then the subclass of the new version
-- is copied from the predecessor version. In the remaining cases, the
-- subclass of the new version is Nil.

-- If Initialize is treated as True, the Predecessor_Update_Id attribute
-- is set to the Last_Update_Id of the source (of the new version's
-- data bits), else Predecessor_Update_Id is Nil.

-- Class specific semantics (these are in addition to or countermand the
-- general rules given above); there are no subclass specific semantics:

-- Class_Directory: The directory will NOT be the root of a new world;
-- must use the World package to do that. The returned Open_Handle will
-- be Nil.  The Initialize parameter is ignored.

-- Class_File: An empty file has organization Binary_File, and contains
-- no data bits.  A non-empty file has the same organization, and data
-- bits.  Only the IO packages should use this operation; "users" should
-- use the Create in the IO packages.

-- Class_Pipe: An empty pipe has some default buffer size, whereas a
-- non-empty pipe has the same buffer size as the predecessor.  The pipe
-- is locked as specified by the Mode, and is open from the point of
-- view of this package.  Note that a mode of Record_Write is typically
-- more useful than the default of Object_Write. Only the IO packages
-- should use this operation; "users" should use the Create in the IO
-- packages.

-- Class_Ada: Additional semantics supplied by the Ada coupler and are as
-- yet unspecified.
```

```
   -- Class_Ada_Attributes: Additional semantics supplied by the Ada coupler
   -- and are as yet unspecified.

   -- Class_Code: Like other classes of objects (such as files), a code
   -- version is modified using the segment operations at the end of this
   -- package. Just as for other classes of objects, the contents of the
   -- version can only be addressed (as a heap) while Open. In addition to
   -- this "heap name", each version of a code object also has a distinct
   -- code segment name assigned to it; the contents of the version can
   -- always be addressed via the code segment name, even when the version is
   -- not Open; but the version cannot be modified via the code segment name.
   -- A page fault on page "I" of the code segment is resolved by creating
   -- the code page I as a copy of the current value of page I+1 of the heap
   -- page (of the corresponding version).

   -- Class_Load_Image: Initialize is always treated as false.

   -- Possible errors:
   --   Is_Bad_Version_Handle (Status):
   --   Is_Bad_Pathname (Status):
   --       o Might be because an object must be created, but the last segment
   --         (name of new child) contains wildcards.
   --   Is_Bad_Action (Status)
   --   Is_Lock_Error (Status):
   --   Predecessor_Version_Is_Not_Known
   --       o Attempting to create a new version of an existing object, and
   --         the Initialize parameter is true, but the selected world view
   --         does not currently select a version of the object, therefore,
   --         we don't know the version number of the "predecessor".
   --   Link_Pack_Entry_Already_Exists
   --       o Attempting to create a new Ada library unit with a simple
   --         name which conflicts with the name of some unit which is
   --         already in the Link_Pack of this world.
   --   Illegal_Class
   --       o This operation is not supported for Class_World_View,
   --         Class_Universe_View, Class_Link_Pack, Class_Tape, or
   --         Class_Terminal.
   --   Object_Class_Mismatch
   --       o Can occur in 1st form only.  If the object already existed,
   --         means that the existing object does not have the class specified
   --         by the non-Nil Class parameter.  If the object did not already
   --         exist, and the class param and "'C" attribute are both specified
   --         and neither is Nil, then this means that they are not equal.
   --   Object_Index_Already_In_Use
   --       o The Index parameter is non-Nil and refers to an already
   --         existing object.
   --   world_View_Is_Frozen
   --       o The specified world view is frozen; therefore it is not
   --         possible to create a version of the object.
   --   Is_Resource_Limit_Error (Status):
   --   Is_Version_Policy_Error (Status):
   --       o Undefined_Version_Policy
   --       o Multiple_Versions_Not_Supported
   --             Means that either the class of the existing object does not
   --             support multiple versions, or the
   --             Allow_Explicit_Create_Version bit is not set in the given
   --             policy options. Note that directories can have at most 1
```

```
--              version.
--        o Object_Is_Not_Reserved
--              Attempting to create a new version of the object, but
--              the object's slot (in the target view) does not have the
--              Reserved bit set.


   procedure Open (Name : Om.Unique_Wildcard;
                   Action : Om.Action_Id;
                   Mode : Om.Lock_Mode;
                   Handle : out Open_Handle;
                   Status : out Error.Condition;
                   Class : Om.Class_Number := Om.Nil;
                   In_Context : Context := Job.Default_Context;
                   Ignore_Reserved_Bit : Boolean := False;
                   Max_Wait : Duration := Job.Default_Wait);

   procedure Open (The_Version : Om.Version_Handle;
                   Action : Om.Action_Id;
                   Mode : Om.Lock_Mode;
                   Handle : out Open_Handle;
                   Status : out Error.Condition;
                   Ignore_Reserved_Bit : Boolean := False;
                   Max_Wait : Duration := Job.Default_Wait);

   -- Usual rules for Naming.Resolve, and interpretation of the Version_Handle.

   -- The following special rules take precedence over the usual rules:

   -- If  (a) The_Version.Version.Number specifies a Nil version and
   --     (b) Mode is one of the update modes (eg, Write_Object) and
   --     (c) The selected world view is not frozen and
   --     (d) Either
   --           (1) The Private_Version_On_Every_Update policy bit
   --               is set, or
   --           (2) The Private_Version_On_First_Update policy bit
   --               is set, and the version that would be selected as a
   --               target under the normal rules is either frozen or
   --               referenced more than once
   -- then an Open_Handle is generated by an implicit call to Create, with
   -- Initialize = True; this is known as "differentiation". If the
   -- Implicit_Versions_Replace_Current policy bit is set, the management of
   -- the retention list is slightly different (than the rules specified for
   -- the Create procedure). Specifically, the new version simply replaces
   -- the current version.

   -- If  (a) The_Version.Version.Number is not Nil and
   --     (b) Mode is one of the update modes (eg, Write_Object) and
   --     (c) The Prevent_Update_Without_View policy bit is not set and
   --     (d) The specified version is not frozen
   -- then the identified version is open'd for update. Note that this rule
   -- will allow the following: One can update an unfrozen version which is
   -- shared by more than one view. One can update a deleted (but not yet
   -- expunged) version.

   -- If The_Version.Version.Number is Nil and the Mode is not in the set
   -- {Unsynchronized, Supercedeable_Read}, then the action acquires a
   -- Record_Read lock on the slot; in all other cases, the given action
```

```
-- acquires no slot lock. In any case, the action acquires the requested
-- lock on the selected version. Note that if differentiation results, a
-- seperate action is created which acquires a slot Record_Write lock, and
-- Object_Read lock on the copied (predecessor) version, and is committed.

-- Due to internal implementation requirements, when a new version is
-- implicitly created (as defined by the above paragraph), the
-- undifferentiation is actually done on a separate action which is
-- committed immediately; thus, the undifferentiation will not be undone
-- by action abandon. This should be "transparent".

-- when a new version is implicitly created, the new version inherits
-- the subclass attribute from the source (of the new version's data bits).

-- When a new version is implicitly created, the Predecessor_Update_Id
-- attribute is set to the Last_Update_Id of the source (of the new
-- version's data bits), else Predecessor_Update_Id is Nil.

-- When an object is first Open'd (for Supersedable read), and later
-- Open'd for update, the Prior_Update_Id function should be used to
-- detect a situation in which the object has been modified between the
-- initial Open (for read) and the subsequent Open for update.

-- To find out what version was Open'd, feed the Open_Handle to the Version
-- function.

-- Class specific semantics:

-- Class_Code: Observe that if Mode is one of the update modes, but
-- differentiation was not performed, this operation gives one the
-- ability to change the contents of an existing code segment.
-- Remember that code segment names are assigned to new versions, not
-- new generations. See the Refresh_Code_Segment operation.

-- Class_Load_Image: Update modes are not allowed in conjunction with
-- objects of this class. One must use the Create operation and get a
-- new version. This restriction might be relaxed at some future date.

-- Possible errors:
--   Is_Bad_Version_Handle (Status)
--   Is_Bad_Pathname (Status)
--   Is_Bad_Action (Status)
--   Is_Lock_Error (Status):
--   Illegal_Class
--       o This operation is not supported for Class_Directory,
--         Class_Universe_View, Class_World_View, and Class_Link_Pack.
--   Version_Is_Frozen
--   Is_Resource_Limit_Error (Status):
--   Is_Version_Policy_Error (Status):
--       o Object_Is_Not_Reserved
--             Attempted to Open with one of the update modes,
--             The_Version.Version.Number is Nil, the
--             Ignore_Reserved_Bit parameter is False, and the
--             the object's slot (in the target view) does not have the
--             Reserved bit set.
--       o Must_Update_Via_View
--             The_Version.Version.Number is not Nil, but the
--             Prevent_Update_Without_View policy bit is set.
```

```
    procedure Delete (Name : Om.Unique_Wildcard;
                      Action : Om.Action_Id;
                      Status : out Error.Condition;
                      In_Context : Context := Job.Default_Context;
                      Max_Wait : Duration := Job.Default_Wait);

    procedure Delete (The_Object : Om.Object_Handle;
                      Action : Om.Action_Id;
                      Status : out Error.Condition;
                      Max_Wait : Duration := Job.Default_Wait);

    -- The usual rules for Naming.Resolve. Usual rules for selecting a
    -- world view (from The_Object).

    -- The following additional rules apply: Object.Version.Number is ignored.
    -- Nil is pushed onto the retention list (of the selected view),
    -- following the usual rules for creation of a new version (the new
    -- version simply happens to be Nil).

    -- Note that the "deleted" version(s) can still be named (using string
    -- names) via other referencing world views, or by use of the "'V"
    -- attribute; and they can, of course, be named by their Version_Id's.

    -- If the deleted version has Class_Ada and a parent which is not
    -- Class_Ada, then the corresponding Link_Pack entry is deleted, with the
    -- usual rules for undifferentiating the Link_Pack. There are additional
    -- (unspecified) obsolecense rules for Class_Ada, Class_Ada_Attributes, and
    -- Class_Code.

    -- The action will acquire a Record_Write lock on the corresponding slot.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --       o Object_Not_In_View
    --             o The object is already deleted in this view.
    --   Is_Bad_Pathname (Status)
    --   Is_Bad_Action (Status)
    --   Is_Lock_Error (Status):
    --   Illegal_Class
    --       o This operation is not supported for Class_Universe_View,
    --         Class_World_View, and Class_Link_Pack.
    --   World_View_Is_Frozen
    --   Is_Resource_Limit_Error (Status)
    --   Is_Version_Policy_Error (Status):
    --       o Object_Is_Not_Reserved
    --             The object's slot (in the target view) does not have the
    --             Reserved bit set.


    generic
        with procedure Visit (Version_To_Be_Deleted : Om.Version_Handle);
        with procedure Handle_Resolve_Error (Status : Error.Condition);
        with procedure Handle_Delete_Error (Status : Error.Condition;
                                            bad_Version : Om.Object_Handle);

    procedure Wild_Delete (Name : Om.Full_Wildcard;
```

```
                        Action : Om.Action_Id;
                        In_Context : Context := Job.Default_Context;
                        Max_Wait : Duration := Job.Default_Wait);

    -- Basically, employs Naming.Wild_Resolve (using Name, Action, and
    -- In_Context) to produce a sequence of Version_Handle's; errors produced
    -- by Wild_Resolve are fed to Handle_Resolve_Error.  For each
    -- Version_Handle produced by Wild_Resolve, calls Visit (with a fully
    -- resolved Version_Handle), and then the corresponding form of Delete;
    -- its errors are fed to Handle_Delete_Error.

    -- As usual, the caller is free to raise arbitrary exceptions in the
    -- generic formal subprograms; these terminate the deletion process,
    -- however, it is up to the client to abandon the action, if that is what
    -- is desired.

    -- Note that one can use "...Foo'V(@)" to cause the object to be deleted
    -- from all views.

    -- Note that one can do things like: set Max_Wait to 0, and ignore
    -- calls to Handle_Delete_Error which produce Status for which
    -- Is_Lock_Error is true.

    -- Clearly, this operation can do lots of damage, especially when followed
    -- by World.Expunge.

    -- RESTRICTION: not supported in 1st qtr impl.

    -- Possible errors:
    --      o Union of those produced by Naming.Wild_Resolve and the
    --        Version_Handle form of Delete.


    procedure Undelete (Name : Om.Unique_Wildcard;
                        Action : Om.Action_Id;
                        Status : out Error.Condition;
                        In_Context : Context := Job.Default_Context;
                        Max_wait : Duration := Job.Default_Wait);

    procedure Undelete (The_Version : Om.Version_Handle;
                        Action : Om.Action_Id;
                        Status : out Error.Condition;
                        Max_Wait : Duration := Job.Default_Wait);

    -- Usual rules for Naming.Resolve.

    -- The version to be undeleted (and the view from which it is undeleted) is
    -- chosen by the normal rules for the interpretation of a source
    -- Object_Handle.

    -- The following additional rules apply:
    -- If The_Version.Version.Number = Nil then
    --     If the current version is Nil then
    --         pop the Nil off the top of the retention list;
    --         causing the acquisition of a Record_Write lock on the slot;
    --     else
    --         the operation fails;
    --     end if;
```

```
    -- else
    --     If it refers to a version in the retention list then
    --         the identified version is removed from the retention list, and
    --         pushed on at the top of the list, becoming current, and
    --         causing the acquisition of a Record_Write lock on the slot;
    --     else
    --         the operation fails;
    --     end if;
    -- end if;

    -- When Nil is popped off, a Link_Pack entry may be created, according to
    -- the usual rules for creation of new versions.

    -- There are additional (unspecified) obsolecense rules for Class_Ada,
    -- Class_Ada_Attributes, and Class_Code.

    -- The action will acquire a Record_Write lock on the corresponding slot.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --       o Object_Not_In_View
    --             o Either the specified version is not in the retention list,
    --               or no version was specified and the retention list
    --               contains all Nils.
    --   Is_Bad_Pathname (Status)
    --   Is_Bad_Action (Status)
    --   Is_Lock_Error (Status)
    --   Illegal_Class
    --       o This operation is not supported for Class_Universe_View,
    --         Class_World_View, and Class_Link_Pack.
    --   World_View_Is_Frozen
    --   Is_Resource_Limit_Error (Status)
    --   Is_Version_Policy_Error (Status):
    --       o Object_Is_Not_Reserved
    --             The object's slot (in the target view) does not have the
    --             Reserved bit set.


    procedure Expunge (Name : Om.Full_Wildcard;
                       Number_Expunged : out Natural;
                       Disk_Blocks_Freed : out Natural;
                       Status : out Error.Condition;
                       In_Context : Context := Job.Default_Context;
                       Violate_Safeguards : Boolean := False);

    procedure Expunge (The_Version : Om.Version_Handle;
                       Number_Expunged : out Natural;
                       Disk_Blocks_Freed : out Natural;
                       Status : out Error.Condition;
                       Violate_Safeguards : Boolean := False);

    -- The meaning of the word "expunge" is defined by the semantics of this
    -- operation. Note that these semantics apply to all classes of object,
    -- including interesting cases like universe views, world views, and
    -- directories.

    -- Note that this is the only operation for destroying universe views,
    -- and therefore the only operation for removal of unwanted releases, etc.
```

```
    -- Note that one can protect frozen views from expunge via access control.

    -- Usual rules for Naming.Resolve and interpretation of Version_Handle.
    -- Plus the following:

    -- If The_Version.Version.Number is Nil then
    --    remove all deleted versions in the retention list
    --        (by "deleted", we mean one which is not current),
    --    causing the acquisition of a Record_Write lock on the slot;
    -- else
    --    remove the specific version from the retention list, if its there,
    --    causing the acquisition of a Record_Write lock on the slot;
    -- end if;

    -- Note that the above rule is simply a noop in the case where the
    -- selected universe view does not reference a world view, or the slot
    -- cannot be locked.

    -- If any versions removed (from the retention list) in the above step are
    -- expungeable, then reclaim their disk space.

    -- If (at this point) the object itself is expungeable, then it ceases to
    -- exist. Note that if the object itself is not expungeable, this
    -- operation may leave the object in existence, without any versions.

    -- If the object itself is expunged, and the object's parent is
    -- expungeable, then the parent is also expunged. This rule is applied
    -- recursively.

    -- A version is physically expungeable if it is not open (in a mode
    -- other than Unsynchronized).

    -- A version's expunge safeguards are satisified if the version is not
    --   (1) Referenced by a world view; and not
    --   (2) A world view referenced by a universe view in the
    --       same world; and not
    --   (3) A world view protected by a still existing universe view
    --       in a potentially different world; and not
    --   (4) A secondary load image referenced by a still existing
    --       primary; and not
    --   (5) An Ada unit referenced by a load image; and not
    --   (6) A code segment referenced by a load image.

    -- A version is expungeable if
    --   (1) It is physically expungeable; and
    --   (2) Either the version's expunge safeguards are satisfied, or the
    --       Violate_Safeguards parameter is true.

    -- An object is expungeable if
    --   (1) There are 0 versions of the object; and
    --   (2) The object has no children.

    -- In the event that an object/version is expunged even though the
    -- safeguard conditions are not met, the operation does NOT attempt to
    -- first remove the references, rather it leaves the references
    -- "dangling"; this behaviour is desirable in cases where the references
    -- come from damaged or mysteriously vanished objects. Generally speaking,
    -- this option is used only as a "last resort". It might be used to
```

```
    -- recover from objects which have been damaged by software bugs or disk
    -- errors.

    -- Note that this operation does not follow action semantics.

    -- On return, Number_Expunged indicates how many versions this operation
    -- actually expunged, and Disk_Blocks_Freed indicates how many 1K byte
    -- blocks were freed.

    -- Class specific semantics:

    -- Class_Directory (subclass World): The single version of the world
    -- is not expunged by this operation. Use World.Destroy instead.

    -- Class_Load_Image: If any of the referenced Ada units or code segments
    -- are now no longer referenced, then the referenced version (and possibly
    -- object) is expunged. For a primary load image, expunges all of the
    -- referenced secondary load image versions (and probably objects).

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --   Is_Resource_Limit_Error (Status)
    --   World_View_Is_Frozen
    --   Illegal_Class
    --       o This operation is not supported for Class_Universe_View,
    --         Class_World_View, and Class_Link_Pack.
    --   Is_Version_Policy_Error (Status):
    --       o Object_Is_Not_Reserved
    --             The object's slot (in the target view) does not have the
    --             Reserved bit set. This will only result when one attempts to
    --             effect the "current" version.


    procedure Copy (Source : Om.Unique_Wildcard;
                    Destination : Om.Unique_Wildcard;
                    Action : Om.Action_Id;
                    handle : out Open_Handle;
                    Status : out Error.Condition;
                    Source_Context : Context := Job.Default_Context;
                    Destination_Context : Context := Job.Default_Context;
                    Max_Wait : Duration := Job.Default_Wait);

    procedure Copy (Source : Om.Object_Handle;
                    Destination : Om.Object_Handle;
                    Action : Om.Action_Id;
                    Handle : out Open_Handle;
                    Status : out Error.Condition;
                    Max_wait : Duration := Job.Default_Wait);

    -- Usual rules for Naming.Resolve and interpretation of Object_Handle's.

    -- Open the source basically as follows:
    --    Object.Open (Source, Action, Mode => Read_Object);
    -- This acquires the usual slot and version locks, as specified by Open.

    -- If  (a) Source and Target refer to the same object; and
    --     (b) Either the Private_Version_On_First_Update or
    --         Private_Version_On_Every_Update policy bits are set
```

```
    -- then the target view is modified to reference the source version, as
    -- follows. If the object is deleted in the target view, or Open for
    -- update would create and push a new version, then the version is pushed
    -- onto the target retention list; otherwise, the version simply replaces
    -- current in the target retention list.

    -- Note that above capability can be used to construct arbitrary world views
,
    -- and can therefore be used to merge paths, etc, by building the
    -- appropriate differential view.

    -- In the remaining cases, we actually copy data. If the object is deleted
    -- in the target view, then we open the target as follows:
    --    Create (Destination, Action, Initialize => False);
    -- else:
    --    Open (Destination, Action, Mode => Write_Object);
    -- The data bits of the source are then copied into the target.  The
    -- virtual length of the target becoming the same as the virtual length of
    -- the source.

    -- The target version is left open, by the returned Open_Handle.

    -- Works for Class_Load_Image; in the case of a physical copy, does the
    -- "right thing" with respect to reference counts on the referenced
    -- Ada and code versions.

    -- Possible errors:
    --       o Union of those produced by Open and Create.
    --       o Illegal_Class
    --             o This operation is not supported for Class_Directory,
    --               Class_Universe_View, Class_World_View, Class_Link_Pack,
    --               Class_Tape, and Class_Terminal.
    --       o Object_Class_Mismatch
    --             o If the destination object already exists, it must have the
    --               same class as the source.
    --    Is_Version_Policy_Error (Status):
    --        o Object_Is_Not_Reserved
    --             o Can result from attempting to copy by reference in a slot
    --               (in the destination view) which does not have the
    --               Reserved bit set. Can result from the implicit calls to
    --               Create or Open.


    generic
        with procedure Visit (Source : Om.Version_Handle;
                              Destination : Om.Version_Handle);
        with procedure Handle_Resolve_Error (Status : Error.Condition);
        with procedure Handle_Substitution_Error
                            (Status : Error.Condition;
                             Bad_Version : Om.Version_Handle);
        with procedure Handle_Copy_Error
                            (Status : Error.Condition;
                             Attempted_Source : Om.Pathname_String;
                             Attempted_Target : Om.Pathname_String);

    procedure Wild_Copy (Source : Om.Full_Wildcard;
                         Destination : Om.Full_Wildcard;
                         Action : Om.Action_Id;
```

```
                        Source_Context : Context := Job.Default_Context;
                        Destination_Context : Context := Job.Default_Context;
                        Max_Wait : Duration := Job.Default_Wait);

    -- Employs Naming.Wild_Resolve (using Source, Action, and Source_Context)
    -- followed by Object.Resolve to produce a sequence of Version_Handle's;
    -- errors are fed to Handle_Resolve_Error.  For each Version_Handle produced
    -- by resolve, employs Naming.Substitute (using the generated
    -- Version_Handle, its pathname, and the Destination substitution pattern)
    -- to produce the target name; errors are fed to
    -- Handle_Substitution_Error.  Calls the string form of copy for each
    -- generated pair of pathnames (using the full pathnames, Action, and
    -- Option); errors are fed to Handle_Copy_Error; calls Visit prior to each
    -- call to Copy.

    -- Properly handles creation of directories in the target context.

    -- What does it do with source universe and world views?

    -- RESTRICTION: not supported in 1st qtr impl.

    -- Possible errors:
    --       o Union of those produced by Naming.Wild_Resolve,
    --         Naming.Substitute, Open, Create and Copy.


    procedure Refresh_Code_Segment (Handle : in out Open_Handle;
                                    Status : out Error.Condition);

    -- Recall that Open will allow one to modify an existing code segment.
    -- Recall that code pages will lay around in the cache until reclaimed
    -- by the page replacement policy. Thus, in order to see the changes
    -- (that have been made in the heap segment) via the code segment, one
    -- must first employ this operation.

    -- Clearly this operation is a waste of time when applied to a new
    -- version of the code object.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --       o Version (Handle) does not exist - probably a bad handle.

    --------------------------------
    -- Contents of an Open_Handle --
    --------------------------------

    -- The functions in this section will return the closest thing to Nil when
    -- given bad parameters or other errors occur.

    function Version (Handle : Open_Handle) return Om.Version_Handle;

    -- Identifies the object, version and universe in which it was
    -- Create'd/Open'd. If Create/Open created a new version, identifies the
    -- newly created version.

    function Predecessor_Version (Handle : Open_Handle)
                  return Om.Version_handle;
```

```
    -- If the Create/Open created a new version, then this function returns
    -- the identity of the version from which the new version was created,
    -- else Nil. Can be used to detect when the view is differentiated.

    function Segment (Handle : Open_Handle) return Machine.Segment_Name;

    -- Identifies the memory addresses which will be used to manipulate the
    -- contents of the open version.  Can use the Pointer_To_First_User_Bit
    -- function (in the segment operations, below) to get a typed pointer
    -- to the data in the segment.

    function Prior_Update_Id (Handle : Open_Handle) return Om.Unique_Id;

    -- Returns the value of Last_Update_Id as it existed just prior to the
    -- Open, or Nil when the Open created a new version. This information can
    -- be used to detect that the contents of an object have changed between
    -- an Open for read (perhaps supersedeable) and later Open/Create. A
    -- common application is to validate an editor image at first edit.

    function Lock_Handle (Handle : Open_Handle)
                  return Action.Object_Lock_Handle;
    function Version_Map_Pointer (Handle : Open_Handle)
                  return Machine.Address;

    -- These operations are restricted to internal OM and other "registered"
    -- clients; specifically these are not available to user programs.

    ----------------------------------------
    -- Read only object/version attributes --
    ----------------------------------------

    -- The functions in this section will return the closest thing to Nil when
    -- given bad parameters or other errors occur. The procedures will be
    -- noops when given bad parameters or other errors occur.

    -- The procedures all take Open_Handle's, which must have Write_Object
    -- access to the desired version, else the operation is a noop.

    -- Note that if one applies a Get_xxx operation to a version which happens
    -- to be open for update, one will receive the values which are current
    -- for the open generation.

    function Exists (The_Version : Om.Version_Id) return Boolean;

    -- If The_Version.Number is Nil, returns true iff the specified object
    -- exists, else returns true iff the specified version exists. This is
    -- one of the few operations which will take Version_Id's for job
    -- worlds and segments.

    function Exists (The_Version : Om.Version_Handle) return Boolean;

    -- Returns true iff the specified version is the "current" version in the
    -- specified universe. If the version number component of The_Version is
    -- Nil, this is equivalent to asking if the specified object is deleted
    -- (from the point of view of the given universe).

    function Is_Frozen (The_Version : Om.Version_Handle) return Boolean;
```

```
    -- True iff the specified version exists and is either a frozen universe
    -- view, a frozen world view, or is referenced by one or more frozen world
    -- views.

    function Version_Count (The_Object : Om.Object_Handle) return Positive;

    -- Returns the number of existing versions of the object.

    function Version_Policy (The_Version : Om.Version_Handle)
                  return Om.Version_Control_Policy;

    function Predecessor_Update_Id (The_Version : Om.Version_Handle)
                  return Unique_Id;

    -- Will return the Last_Update_Id of the previous version whose data bits
    -- were copied into this version when it was created. Very subtle note:
    -- when a write creates a new version to carry out its portion of the
    -- Supersedeable_Read_Object protocol, the new version keeps the same
    -- Predecessor_Update_Id value.

    -- This information can be used by the configuration management tools to
    -- detect a situation in which a sublibrary is being reintegrated into its
    -- parent library, and the parent's unit has been recompiled, which might
    -- force the sublibraries's unit to be reintegrated in the source state.

    function Creation_Id (Version : Om.Version_Handle)
                  return Om.Net_Unique_Id;

    -- Returns the network wide creation timestamp for the identified version.
    -- This is one of the few operations which will take Version_Id's for job
    -- worlds and segments.

    function Last_Update_Id (Version : Om.Version_Handle) return Unique_Id;
    function Last_Update_User (Handle : Open_Handle)
                  return Om_Definitions.User_Id;

    function Last_Access_Time
                  (Version : Om.Version_Handle) return Calendar.Time;
    -- RESTRICTION: not supported in 1st qtr impl.

    function Creating_User (Version : Om.Version_Handle)
                  return Om_Definitions.User_Number;

    function Disk_Blocks_Consumed
                  (Version : Om.Version_Handle) return Integer;

    -- Number of 1K byte disk blocks actually consumed by the version itself.
    -- This is one of the few operations which will take Version_Id's for job
    -- worlds and segments.

    function Virtual_Page_Length (Version : Om.Version_Handle) return Integer;

    -- Assuming no holes, indicates the number of pages which can can be
    -- addressed (without getting Nonexistent_Page_Error). This is one of the
    -- few operations which will take Version_Id's for job worlds and segments.

    function User_Bit_Length (Version : Om.Version_Handle) return Integer;
```

```
    -- Number of user bits stored in the version. Does not include ucode/OM
    -- overhead.

    function Disk_Blocks_Reclaimable_By_Expunge
                  (Version : Om.Version_Handle) return Integer;

    -- Number of 1K byte disk blocks which would be reclaimed if the
    -- version was expunged. For normal objects, same result as the
    -- Disk_Blocks_Consumed function. But for universe views, world views,
    -- and load images, includes the space that would be reclaimable as
    -- a result of fewer references. This is one of the few operations which
    -- will take Version_Id's for job worlds and segments.

    ------------------------------------
    -- Read/Write version attributes --
    ------------------------------------

    -- The functions in this section will return the closest thing to Nil when
    -- given bad parameters or other errors occur. The procedures will be
    -- noops when given bad parameters or other errors occur.

    -- Most procedures take Open_Handle's, which must have Write_Object
    -- access to the desired version, else the operation is a noop.

    -- Note that if one applies a Get_xxx operation to a version which happens
    -- to be open for update, one will receive the values which are current
    -- for the open generation.

    function Get_Subclass (Version : Om.Version_Handle)
                  return Om.Subclass_Number;
    procedure Set_Subclass (Handle : in out Open_Handle;
                            New_Subclass : Om.Subclass_Number);

    function Acl (Version : Om.Version_Handle) return Om.Access_Control_List;
    procedure Set_Acl (Handle : Om.Version_Handle;
                       New_Acl : Om.Access_Control_List;
                       Status : out Error.Condition);
    -- RESTRICTION: not supported in 1st qtr impl.

    function Reserved (Version : Om.Version_Handle) return Boolean;
    procedure Set_Reserved (Handle : in out Open_Handle;
                            New_Version : Boolean);
    -- Used by configuration mgmt tools to control access to objects.


    function Code_Name (Version : Om.Version_Handle) return Machine.Module_Name;
    procedure Set_Code_Name (Handle : in out Open_Handle;
                             New_Name : Machine.Module_Name;
                             Status : out Error.Condition);

    -- The procedure can only be used for MV generated code segs and requires
    -- update access to every load image which references the open version.


    -- Possible errors:
    --   Illegal_Class
    --       o Can only be applied to objects with Class_Code.
    --   Not_A_Cross_Cg_Segment
```

```
--         o World (New_Name) must be in Machine.Cross_Cg_Worlds.
--    Not_In_Cross_Cg_Object_World
--         o The associated world (of the given object) must be that
--           associated with "!Machine".
--    Not_A_New_Code_Segment
--         o This must be a new version of the object not referenced by any
--           load image.

function Last_Update_Time (Version : Om.Version_Handle)
              return Calendar.Time;
procedure Set_Last_Update_Time (Handle : in out Open_Handle;
                                Update_Time : Calendar.Time);

function Want_Backup (Version : Om.Version_Handle) return Boolean;
procedure Set_Want_Backup (Handle : in out Open_Handle;
                           New_Value : Boolean);

-- Versions are backed up iff the Want_Backup flag is set. The default
-- is Want_Backup => true.

-----------------------------------------------
-- Ada specific read/write version attributes --
-----------------------------------------------

-- The functions in this section will return the closest thing to Nil when
-- given bad parameters are other errors occur. The functions do NOT
-- get action locks, thus garbage results are theoretically possible.
-- The procedures will get a Record_Write lock on the attribute.

function Unit_State (Version : Om.Object_Handle) return Om.Ada_Unit_State;

generic
    with procedure Make_Change (Unit_State : in out Om.Ada_Unit_State);
procedure Set_Unit_State (Version : Om.Object_Handle;
                          Status : out Error.Condition);

-- The function returns the current state of the identified unit.
-- The procedure can make arbitrary changes to the unit state, via the
-- supplied Make_Change operation, acquiring a Record_Write lock on
-- the unit state of the identified unit.


function Depends_On (Object : Om.Object_Handle)
              return Om.Dependency_Vector;

generic
    with procedure Make_Change (Depends_On : in out Om.Dependency_Vector);
procedure Set_Depends_On (Object : Om.Object_Handle;
                          Status : out Error.Condition);

-- The function returns the current value of the identified row of the
-- Dependency_Matrix. The procedure can make arbitrary changes to the row,
-- via the supplied Make_Change operation, acquiring a Record_Write lock on
-- the row for the identified unit.

function Dependents (Object : Om.Object_Handle)
              return Om.Dependency_Vector;
```

```
    -- The function returns the current value of the identified column of the
    -- Dependency_Matrix. There is no operation to change a column of the
    -- matrix, without resorting to the change row operation (above).

    -------------------------------
    -- Relations between objects --
    -------------------------------

    generic
        with procedure Visit (The_Version : Om.Version_Handle);
    procedure Traverse_Versions (Of_Object : Om.Object_Handle;
                                 Status : out Error.Condition);

    -- If Of_Object.Universe is non-Nil and it selects a world view, then this
    -- operation calls Visit for every version Of_Object in the specified
    -- universe. If the slot (in the world view) has non-Nil entries, Visit
    -- will be called for trailing Nil's. For example, if the retention list
    -- is <nil, nil, v1, v4, v7, nil> (the versions are listed in the order
    -- that they were pushed), Visit will be called 4 times, for v1, v4, v7
    -- and nil, in that order.

    -- Otherwise (.Universe is Nil, or doesn't select a world view), calls
    -- Visit for every version Of_Object, regardless of what views they
    -- participate in. Visit is called in sort order, by Version_Number.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --      o Something wrong with Of_Object


    generic
        with procedure Visit (Referencing_Version : Om.Version_Handle);
        with procedure Handle_Error (Status : Error.Condition);

    procedure Locate_Expunge_Preventing_References
         (The_Version : Om.Version_Handle;
          Status : out Error.Condition);

    -- The operation finds the expunge-preventing references to the specified
    -- version of the object.

    -- Calls Visit for every expunge-preventing reference. Depending upon the
    -- class of The_Object, Referencing_Version may refer to a universe view,
    -- world view, or load image. It is possible that some of the information
    -- may not be available due to locks held by other actions; calls
    -- Handle_Error in these cases. As usual, the caller is free to raise
    -- arbitrary exceptions in the generic formal subprograms; these terminate
    -- the search.

    -- This operation is useful for determining why a particular version fails
    -- to be expunged.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --   Is_Lock_Error (Status);
    --   Is_Resource_Limit_Error (Status)
```

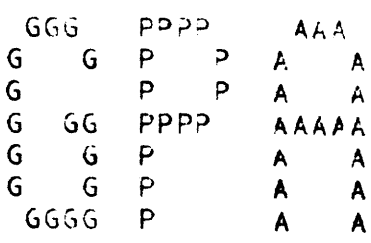
```
    generic
        with procedure Visit (The_Object : Om.Object_Handle);

    procedure Locate_Objects_With_Given_Class
         (In_World : Om.Version_Handle;
          Class : Om.Class_Number;
          Status : out Error.Condition);

    -- The world number component of In_World is used to identify the
    -- world to be searched.  Calls Visit for every object (in the specified
    -- world) which has the specified Class.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --   Illegal_Class


    generic
        with procedure Visit (The_Universe_View : Om.Version_Handle);

    procedure Locate_Referencing_Universes
         (The_World_View : Om.Version_Handle;
          Status : out Error.Condition);

    -- Looks through all worlds (on this machine), calling Visit for every
    -- universe view which references the specified world view version.

    -- Possible errors:
    --   Is_Bad_Version_Handle (Status)
    --   Illegal_Class

    ------------------------------------------------
    -- Class and Subclass names, and registration --
    ------------------------------------------------

    function To_String (Class : Om.Class_Number) return String;

    -- Serves as the 'Image function for Om.Class_Number. Returns "<undefined>"
    -- for bogus input.

    function To_String (Class : Om.Class_Number;
                        Subclass : Om.Subclass_Number) return String;

    -- Serves as the 'Image function for Om.Subclass_Number. Note that subclass
    -- numbers are defined in the context of a particular class. Returns
    -- "<undefined>" for bogus input.

    package Directory_Subclass is
        function World return Om.Subclass_Number;
        function Path return Om.Subclass_Number;
        function Session return Om.Subclass_Number;

    end Directory_Subclass;
    --pragma Integrate (Directory_Subclass);

    package File_Subclass is
        -- The following subclasses are predefined:
        function Binary return Om.Subclass_Number;
```

```
        function Text return Om.Subclass_Number;
        function Message_Window return Om.Subclass_Number;
        function Text_Io_Window return Om.Subclass_Number;
        function Switches return Om.Subclass_Number;
        function Editor_Macros return Om.Subclass_Number;
    end File_Subclass;
    --pragma Integrate (File_Subclass);

    package Ada_Attribute_Subclass is
        -- The following subclasses are predefined:
        function Compatibility_Database return Om.Subclass_Number;
        function Ada_Image return Om.Subclass_Number;
        function Cg_Attributes return Om.Subclass_Number;
    end Ada_Attribute_Subclass;
    --pragma Integrate (Ada_Attribute_Subclass);

    procedure Define_Subclass
                    (Class : Om.Class_Number;
                     Subclass : Om.Subclass_Number;
                     Name : String;
                     Policy : Om.Version_Control_Policy := Om.Nil;
                     Status : out Error.Condition);

    -- Used to define a new subclass. If the specified Default_Version_Policy
    -- is Nil, will implicitly use the value returned by the
    -- Default_Version_Policy function, below. The new subclass definition
    -- is now permanent.  Note that the assignment of subclasses is typically
    -- done via a manual registration process, at Rational.

    -- Possible errors:
    --   Object_Class_Does_Not_Exist
    --   Object_SubClass_Already_Exists
    --       o The given Subclass and Name may not previously exist.
    --   Is_Version_Policy_Error (Status):
    --       o Undefined_Version_Policy

    procedure Undefine_Subclass
                    (Class : Om.Class_Number;
                     Subclass : Om.Subclass_Number;
                     Name : String;
                     Status : out Error.Condition);

    -- Used to delete a previously defined subclass.  Does not verify that
    -- there are no objects which claim to have the now deleted subclass.

    -- Possible errors:
    --   Object_Class_Does_Not_Exist
    --   Object_SubClass_Does_Not_Exist
    --   Object_SubClass_Is_Predefined


    function Default_Version_Policy
                    (Class : Om.Class_Number;
                     Subclass : Om.Subclass_Number := Om.Nil)
                    return Om.Version_Control_Policy;

    -- Returns Standard_Differential_Version_Policy for all predefined
    -- subclasses, including Nil. Exceptions:
```

```
    -- (1) The Compatibility_Database subclass of Ada_Attribute's, uses the
    --     Shared_Version_Policy.
    -- (2) Directory class uses the Shared_Version_Policy.
    -- (3) Universe view class uses the Shared_Version_Policy.
    -- (4) Device classes use the Shared_Version_Policy.

    ------------------------------------------
    -- Segment operations:                  --
    --    Restricted to Rational programs only --
    ------------------------------------------

    -- The following operations assume that one obtained the Segment_Name
    -- argument by using the Segment function on a legitimate Open_Handle.

    -- Segment operations fed bad Segment_Name parameters will raise the
    -- following exceptions:
    --     Illegal_Reference
    --         o The Module_Name portion of the given Segment_Name is 0.
    --     Nonexistent_Page_Error
    --         o The Module_Name portion of the given Segment_Name does not
    --           exist.
    --     Illegal_Heap_Access
    --         o Attempt to address outside the range of "user data bits".


    function Corresponding_Version (The_Segment : Machine.Segment_Name)
                return Om.Version_Handle;

    -- Returns a Version_Handle with a fully resolve Version component, and
    -- nil Universe component. May raise the standard bad Segment_Name
    -- exceptions, as defined above.


    function End_Of_Segment (The_Segment : Machine.Segment_Name)
                return Machine.Bit_Offset;
    function End_Of_Segment (The_Segment : Machine.Segment_Name)
                return Om.Segment_Position;

    -- Returns the identity of the first unused "offset" in the segment.
    -- The first function returns the value in terms appropriate for comparison
    -- with the Bit_Offset component of Machine.Address's.  The second function
    -- returns the value in terms of "user data bits". May raise the standard
    -- bad Segment_Name exceptions, as defined above.


    function Virtual_Length (Of_Segment : Machine.Segment_Name)
                return Long_Integer;
    function Logical_Length (Of_Segment : Machine.Segment_Name)
                return Long_Integer;

    -- The Virtual_Length of a segment is the number of bits between the
    -- first and last legal Bit_Offset's, inclusive.  The Logical_Length of
    -- a segment is the number of bits between the first and last legal
    -- Segment_Position's, inclusive.  By "legal" we mean that if one had
    -- the segment open for update, one could "touch" the addressed bit.
    -- May raise the standard bad Segment_Name exceptions, as defined above.
```

```
    generic
        type Element is limited private;
        type Pointer is access Element;
        pragma Segmented_Heap (Pointer);
    function Get_Segment (P : Pointer) return Machine.Segment_Name;
    -- May raise the standard bad Segment_Name exceptions, as defined above.


    generic
        type Element is limited private;
        type Pointer is access Element;
        pragma Segmented_Heap (Pointer);
    function Get_Position (P : Pointer) return Om.Segment_Position;
    -- May raise the standard bad Segment_Name exceptions, as defined above.

    ----------------------------------------
    -- Segment operations:                --
    --    Extremely dangerous operations --
    --    Restricted to "trusted code"   --
    ----------------------------------------

    generic
        type Element is private;
        type Pointer is access Element;
        pragma Segmented_Heap (Pointer);
    function Pointer_To_First_User_Bit
         (Into : Machine.Segment_Name) return Pointer;

    -- Used to return a typed pointer to the first "user bit" in the segment.


    generic
        type Element is limited private;
        type Pointer is access Element;
        pragma Segmented_Heap (Pointer);
    function Make_Pointer (Into : Machine.Segment_Name;
                           At_Position : Om.Segment_Position) return Pointer;

    -- Can be used to construct pointers from their components. Since one
    -- can make pointers at arbitrary positions, the resulting equivalencing
    -- can be the source of severe damage to the system. One can make a null
    -- pointer by supplying Nil for At_Position. May raise the standard bad
    -- Segment_Name exceptions, as defined above.


    generic
        type Element is limited private;
        type Pointer is access Element;
        pragma Segmented_Heap (Pointer);
    function Convert_To_Pointer (Into : Machine.Address) return Pointer;

    -- Can be used to construct pointers from its address. Since one can make
    -- pointers at arbitrary positions, the resulting equivalencing can be
    -- the source of severe damage to the system.  Will raise
    -- Illegal_Heap_Access if the given address refers to an object with
    -- other than Class_File, Class_Ada, Class_Ada_Attributes, or Class_Code.

    generic
```

```
        type Element is limited private;
        type Pointer is access Element;
        pragma Segmented_Heap (Pointer);
    procedure Assign_Value (Value : Element;
                            Into : Pointer);

    -- For scalars and records without discriminants, this operation is the
    -- same as "Into.all := Value". In particular, if the Element type is
    -- scalar, will raise Contraint_Error and Numeric_Error, as would be
    -- expected from "Into.all := Value".

    -- In the cases where the Element type is an array or record with
    -- discrimants, the semantics are completely different than "Into.all :=
    -- Value"; in this case, the operation does not generate Constraint_Error;
    -- in particular, it simply copies the bits of the source (by looking
    -- inside the source object, to determine 'size) to the memory addressed
    -- by the given pointer; but does not look inside the target (to check
    -- array bounds, discriminants, constraints, etc); thus it simply assumes
    -- that 'size bits have already been allocated at the target address.

    -- May raise the standard bad Segment_Name exceptions defined above.


    procedure Adjust_End_Of_Segment (Of_Heap : Machine.Segment_Name;
                                     New_End_Of_Segment : Om.Segment_Position);

    -- Causes the End_Of_Segment operation to return New_End_Of_Segment. When
    -- New_End_Of_Segment is larger than the current End_Of_Segment, this
    -- operation "extends the heap". When New_End_Of_Segment is smaller than
    -- current End_Of_Segment, this operation "cuts back the heap". May raise
    -- the standard bad Segment_Name exceptions defined above.

private
    type Open_Handle is
        -- For details, see the section containing functions on Open_Handle.
        record
            Version              : Om.Version_Handle;
            Predecessor_Version  : Om.Version_Id;
            Segment              : Machine.Segment_Name;
            Lock_Handle          : Action.Object_Lock_Handle;
            Version_Map_Pointer  : Machine.Address;
        end record;

    Nil_Open_Handle : constant Open_Handle :=
                      (Version => Om.Nil_Object_Handle,
                       Predecessor_Version => Om.Nil_Version_Id,
                       Segment => Machine.Null_Segment_Name,
                       Lock_Handle => Action.Nil_Object_Lock_Handle,
                       Version_Map_Pointer => Machine.Null_Address);

end Object;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.KK.MACHINE
Version: V(14)
Request: 1273

Date: April 24, 1986
Queued: 11:26:36 AM
Printed: 11:39:09 AM

```
with Sys;

package Machine is
    -- Gamma "Machine" package to die. This package is part of the
    -- standard OM export view and contains machine dependent definitions,
    -- mostly addressing conventions.

    -- pragma Subsystem (Kernel);
    -- pragma Module_Name (4, ?);

    package System renames Sys;

    -------------------------
    -- Basic storage units --
    -------------------------

    Bits_Per_Byte              : constant := 8;
    Bits_Per_Word              : constant := 128;

    Bits_Per_Page              : constant := 2 ** 13;
    Bits_Per_Page_Is_Constant  : Boolean := True;

    -- In the future, the first value may be a function and change from
    -- machine to machine, perhaps from boot to boot.  Assumptions to the
    -- contrary should assert the _Is_Constant invariant.

    --=============--
    -- Segments --
    --=============--

    -- Taken together, World_Number, Segment_Number and Segment_Kind uniquely
    -- identify a segment in the machine.  A segment is the fundamental unit
    -- of storage in the virtual memory system.  Kinds Illegal and System are
    -- not used.

    -- Note that in Job_worlds Control, Typ, Queue, and Data segments have the
    -- same lifetime, and are collectively referred to as a module. Import and
    -- Code segments each have distinct lifetimes and are typically referred
    -- to as import spaces and code segments, respectively.

    ------------------
    -- World_Number --
    ------------------

    type World_Number is new Long_Integer range 0..2 ** 10 - 1;

    subtype Kernel_Worlds is World_Number range 0..4;
    -- All Kernel_Layer modules are elaborated in segments in worlds
    -- in this range.

    Ipl_World : constant World_Number := 4;

    subtype Environment_Worlds is World_Number range 4..5;

    -- The module and import spaces of the environment subsystems are
    -- elaborated in these job worlds.  When the virtual memory system is
    -- initialized, the above job worlds are automatically created.
```

```
    subtype Env_Backup_Worlds is World_Number range 6..7;

    -- After a system crash, the stuff in world 4 is "copied" to world 6,
    -- and world 5 "copied" to world 7.  This allows virtual memory to be
    -- started and the system to elaborate prior to dumping the state of
    -- virtual memory associated with the last crash.

    subtype Job_Worlds is World_Number range 8..255;

    -- Jobs run in worlds in this range.  Module amd import names may be
    -- reclaimed when the job terminates and all traces of it have been wiped
    -- from the face of the earth. Until we stop building environment systems
    -- on the MV's, we will not actually give out job numbers in the range
    -- Cross_Cg_Worlds.

    subtype Cross_Cg_Worlds is World_Number range 8..25;

    -- It is still the case that mv generated code segments live on vps 8..25.
    -- These pages are always read_only and clean.  There is NO corresponding
    -- object world.

    subtype Iop_Cross_Cg_Worlds is Cross_Cg_worlds range 8..10;

    -- These are the Cross_Cg_Worlds that are loaded from the IOP.
    -- *** These numbers are not the real ones.  They are yet to be decided on.

    subtype Env_Cross_Cg_Worlds is Cross_Cg_Worlds range 11..25;

    -- These are the Cross_Cg_Worlds that are loaded into EEVB.
    -- *** These numbers are not the real ones.  They are yet to be decided on.

    subtype Task_Worlds is World_Number range 0..255;

    -- Worlds in which the architecture/mts are willing to run tasks.

    subtype Object_Worlds is World_Number range 256..1022;

    -- Segmented heaps (file spaces) live in worlds in this range. These may
    -- be either permanent and temporary.

    Om_World             : constant Object_Worlds := 256;
    Miscellaneous_World  : constant World_Number := 1023;

    -- Om_World contains object management state which is not accounted for by
    -- legitimate directory objects.  There is a corresponding disk world.
    -- Current uses: (a) Action log blocks.

    -- There is no corresponding disk world for Miscellaneous_World. It is
    -- used for addresses of special gook in the memory cache. The following
    -- things are placed in the cache under these addresses: (1) The ucode
    -- assist routines are trapped by doing field executes into this world.
    -- (2) The kernel's page pool pages go in this world. (3) Some of the
    -- Volume_Manager state is mapped via this world.

    -- Beware of the fact that the window of vulnerability mechanism, actions,
    -- and crash recovery all assert the following invariant:
    --      Disk.All_Dirty_World_Info_Mapped_Under_Its_Vp
    -- Moral of the story: be VERY careful with permanent/reconstructed state
```

```
   -- which lives in the cache under these world numbers.

   --------------------
   -- Segment_Number --
   --------------------

   type Segment_Number is new Long_Integer range 0..2 ** 22 - 1;

   subtype Code_Segment_Number is Segment_Number
        range 0..Segment_Number (2 ** 16 - 1);

   First_Allocatable_Segment : constant := 2 ** 6;

   -- For all worlds (both job and file), the first 2**6 segment numbers
   -- are reserved for use as disk mapping pages by Object_Management.

   type Segment_Kind is new Long_Integer range 0..7;

   Illegal_Segment : constant Segment_Kind := 0;
   Control_Segment : constant Segment_Kind := 1;
   Typ_Segment     : constant Segment_Kind := 2;
   Queue_Segment   : constant Segment_Kind := 3;
   Data_Segment    : constant Segment_Kind := 4;
   Import_Segment  : constant Segment_Kind := 5;
   Code_Segment    : constant Segment_Kind := 6;
   System_Segment  : constant Segment_Kind := 7;

   type Module_Name is new Long_Integer range 0..2 ** 32 - 1;

   -- The architecture often knows the Segment_Kind implicitly (from
   -- context) and therefore uses Module_Name to refer to segments.  The
   -- following operations define the representation of a Module_Name.

   function Extract_Segment (Name : Module_Name) return Segment_Number;
   function Extract_World (Name : Module_Name) return World_Number;
   function Construct_Module_Name
                 (Segment : Segment_Number;
                  World : World_Number) return Module_Name;

   --============--
   -- Addresses --
   --============--

   type Bit_Offset is new Long_Integer range 0..2 ** 32 - 1;

   -- Physical bit offset within a particular segment; 0 identifies first
   -- physical bit in the segment.

   subtype Address is System.Address;
   Null_Address : constant Address := System.Null_Address;

   -- Addresses carried around by the machine are of this form:

   -- type Address is
   --      record
   --          Segment : Segment_Number;
   --          World   : World_Number;
   --          Page    : Page_Number;
```

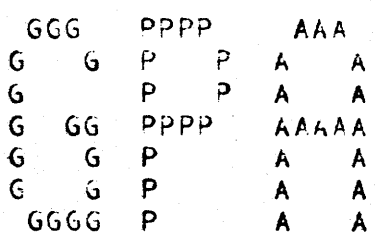
```
--          Byte    : 0 .. 1023;
--          Bit     : 0 .. 7;
--      end record;
-- or, equivalently:
--      record
--          Name    : Module_Name;
--          Offset  : Bit_Offset;
--      end record;

-- Warning: an Address does NOT contain the segment kind; it must be
-- known implicitly.  The following operations define the representation
-- of addresses.

function "<" (L, R : Address) return Boolean;
function "=" (L, R : Address) return Boolean renames System."=";

function Extract_Segment (Addr : Address) return Segment_Number;
function Extract_World (Addr : Address) return World_Number;
function Extract_Name (Addr : Address) return Module_Name;
function Extract_Offset (Addr : Address) return Bit_Offset;
function Construct_Address (Segment : Segment_Number;
                            World : World_Number;
                            Offset : Bit_Offset) return Address;
function Construct_Address (Name : Module_Name;
                            Offset : Bit_Offset) return Address;

type Page_Number is new Long_Integer range 0..2 ** 19 - 1;
function Extract_Page (Addr : Address) return Page_Number;
function Extract_Page (Offset : Bit_Offset) return Page_Number;
function Extract_Bit_Offset_Within_Page (Addr : Address) return Bit_Offset;
function Extract_Bit_Offset_Within_Page
            (Offset : Bit_Offset) return Bit_Offset;

function To_Long_Integer (Addr : Address) return Long_Integer;
function To_Address (Int : Long_Integer) return Address;
-- These are inverses of each other, and preserve bit representation.

subtype Page_Address is Address;
-- Same as an Address, except: bits 51..60 are 0, and bits 61..63 contain
-- the Segment_Kind; ie, it looks like:
--      record
--          Segment : Segment_Number;
--          World   : World_Number;
--          Page    : Page_Number;
--          Byte    : 0 .. 1023; -- all zero
--          Kind    : Segment_Kind; -- in lieu of Bit #
--      end record;

subtype Space_Address is Address;
-- Same as a Page_Address, except: bits 32..50 are 0; ie, it looks like:
--      record
--          Segment : Segment_Number;
--          World   : World_Number;
--          Page    : Page_Number; -- all zero
--          Byte    : 0 .. 1023; -- all zero
--          Kind    : Segment_Kind; -- in lieu of Bit #
--      end record;
-- Unlike the Gamma system, each segment is considered to have its own
```

```
   -- space address.

   function Construct_Space_Address
                (Name : Module_Name;
                 Kind : Segment_Kind) return Space_Address;

   function Construct_Page_Address
                (Name : Module_Name;
                 Page : Page_Number;
                 Kind : Segment_Kind) return Page_Address;

   function To_Data_Address (Addr : Address) return Page_Address;
   function To_Address (Addr : Address;
                        Kind : Segment_Kind) return Page_Address;

   function Extract_Kind (Addr : Space_Address) return Segment_Kind;

   function Change_Kind (Of_Addr : Space_Address;
                         To_Kind : Segment_Kind) return Space_Address;

   function Space (Of_Page : Page_Address) return Space_Address;

   function Change_Page_Number
                (Of_Page : Page_Address;
                 To_Page_Number : Page_Number) return Page_Address;

   type Address_Array is array (Natural range <>) of Address;

   --================================--
   -- Protected segment reference --
   --================================--

   type Segment_Name is private;

   -- Same information content as a Module_Name, but this form has the
   -- bits of the Module_Name left justified, to be in the same position
   -- as in an Address. Also, this type is runtime private in the sense
   -- that the machine will zero its contents when declared and will not
   -- permit aliasing. Thus, one will not find any functions which can
   -- generate this type from integers (or other unprotected types).

   Null_Segment_Name : constant Segment_Name;

   function To_Long_Integer (Seg : Segment_Name) return Long_Integer;
   function To_Module_Name (Seg : Segment_Name) return Module_Name;

   --============================--
   -- Basic Machine Operations --
   --============================--

   subtype Task_Id is Module_Name;
   -- Refers to a module, which can be a package or Ada task.

   function My_Task_Id return Task_Id;
   -- "task id" of the executing thread. Note that packages also have
   -- task ids.

   function Get_Machine_Id return Long_Integer;
```

```
   -- Returns the id for this particular R1000. This value is known to not
   -- exceed 10**6.

   type Cpu_Priority_Level is new Long_Integer range 0..15;

   Best_Cpu_Priority         : constant Cpu_Priority_Level := 0;
   Next_Best_Cpu_Priority    : constant Cpu_Priority_Level := 1;
   Third_Best_Cpu_Priority   : constant Cpu_Priority_Level := 2;
   Fourth_Best_Cpu_Priority  : constant Cpu_Priority_Level := 3;
   Worst_Cpu_Priority        : constant Cpu_Priority_Level := 15;

   -- Note that these notions of priority reflect the internal scheduling
   -- algorithms of the macine. Task priority (per LRM) is defined elsewhere.
   -- Job priority (per MTS) is defined elsewhere.

   function My_Priority return Cpu_Priority_Level;

   -- If called while in rendezvous, will return that of the rendezvous, not
   -- of the task calling the function.

   function I_Am_Wired return Boolean;
   -- Returns true if the executing task's TCB is wired.

   function Is_Callable (The_Task : Task_Id) return Boolean;
   -- Returns the value of the 'Callable attribute (as per LRM).

   package Memory is

      ------------------------------
      -- Memory related operations --
      ------------------------------

      function Number_Of_Sets return Natural;
      function Number_Of_Lines return Natural;
      function Number_Of_Frames return Natural;

      type Cache_Set_Number is new Natural range 0..Number_Of_Sets - 1;
      type Cache_Line_Number is new Natural range 0..Number_Of_Lines - 1;
      type Cache_Frame_Number is new Natural range 0..Number_Of_Frames - 1;

      type Frame_Bits is array (Cache_Frame_Number) of Boolean;

      function Line (Frame : Cache_Frame_Number) return Cache_Line_Number;
      function Set (Frame : Cache_Frame_Number) return Cache_Set_Number;

      function Frame (Line : Cache_Line_Number;
                      Set : Cache_Set_Number) return Cache_Frame_Number;

      procedure Get_Page_Address (In_Frame : Cache_Frame_Number;
                                  Empty : out Boolean;
                                  Contains_Page : out Page_Address);

      -- If the specified cache frame contains no page, then return Empty;
      -- else not Empty and Contains_Page gives the virtual address of the
      -- page in the specified frame.

      function Exists (Page : Page_Address) return Boolean;
```

```
    -- Returns true iff the specified page is currently in the cache.

    function Is_In_Transit (Page : Page_Address) return Boolean;

    -- Returns true iff the specified page exists in the cache and is
    -- in_transit.

    function Is_Dirty (Page : Page_Address) return Boolean;

    -- Returns true iff the specified page exists in the cache and is
    -- marked as dirty.

    function Is_Wired (Page : Page_Address) return Boolean;

    -- Returns true iff the specified page exists in the cache and is
    -- wired.

end Memory;

package Operations is

    --====================================--
    -- Miscellaneous ucode'd operations --
    --====================================--

    type Value_Generator is
        record
            Value : Long_Integer;
        end record;

    procedure Indivisibly_Increment (Generator : in out Value_Generator;
                                     Value : out Long_Integer);

    -- Performs the following:
    --     if Generator.Value < Long_Integer'Last then
    --         Generator.Value := Generator.Value + 1;
    --     end if;
    --     Value := Generator.Value;
    -- This operation is indivisible (atomic), even with respect to page
    -- faults.

    type Boolean_Lock is
        record
            Value : Boolean;
        end record;

    procedure Test_And_Set (The_Lock : in out Boolean_Lock;
                            Value : out Boolean);

    -- Performs the following:
    --     Value := The_Lock.Value;
    --     The_Lock.Value := True;
    -- This operation is indivisible (atomic), even with respect to page
    -- faults.

    procedure Wait_For_Transfer_To_Complete (Page : Page_Address);
```

```
        -- Used to wait for disk transfer completion: Caller should first
        -- issue the appropriate request to the Disk_Driver. Then call this
        -- routine with the page involved in that transfer. If the specified
        -- page does not exist or is not currently in transit, this call
        -- returns immediately; otherwise places the calling task in the disk
        -- wait queue, in which case the call returns when the appropriate
        -- invocation of Roust has been made by Disk_Driver.

    end Operations;

private
    type Segment_Name is access Boolean;
    pragma Segmented_Heap (Segment_Name);
    Null_Segment_Name : constant Segment_Name := null;
end Machine;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.EXECUTION
Version: V(104)
Request: 1274

Date: April 24, 1986
Queued: 11:26:50 AM
Printed: 11:39:41 AM

```
with Job;
with Error;
with Object;
with Machine;
with Om_Definitions;

pragma Private_Eyes_Only;
with Execution_Layout;
with Load_Image_Layout;

package Execution is
    -- pragma Subsystem (Object_Management);
    -- pragma Module_Name (4, ?);

    -- Entities to be executed include main programs and subsystems, both are
    -- represented by load image objects (produced by the "loader"). The
    -- contents of a load image object is defined by the Load_Image package
    -- (below). There are various operations which cause the execution
    -- (elaboration) of load images (representing main programs and
    -- subsystems). These are defined in the Operations package (below).

    package Om renames Om_Definitions;

    subtype Compatability_Vector is Om.Compatability_Vector;
    subtype Code_Segment_Key is Natural range 0..2 ** 16 - 1;
    subtype Secondary_Number is Natural range 0..2 ** 16 - 1;
    subtype Imported_Unit_Number is Natural range 0..2 ** 16 - 1;
    subtype Imported_Subsystem_Number is Natural range 0..2 ** 16 - 1;
    subtype Imported_Program_Number is Natural range 0..2 ** 16 - 1;
    subtype Exported_Module_Position is Natural range 0..2 ** 16 - 1;
    subtype Imported_Module_Position is Natural range 0..2 ** 16 - 1;
    subtype Elaboration_Order_Array is Om.Simple_Object_Id_Array;
    subtype Exception_Information is Om.Exception_Information;

    subtype Root_Of_World is Om.Object_Handle;
    subtype Some_Object_In_World is Om.Object_Handle;

    ----------------
    -- Load images --
    ----------------

    package Load_Image is

        -- Recall the definition of "partial order defined by unit with
        -- clauses" in section 10.3 of the LRM. We use the term "unit with
        -- closure" to mean the LRM's partial order, treating the entire
        -- directory system as one big Ada library.

        -- Here we use the term "main program" to mean a library unit procedure
        -- which is (to be) loaded as a main program.

        -- Here we use the term "subsystem" to mean a world which is (to be)
        -- loaded as a subsystem (in the current view).

        -- One job of the loader is to generate code to cause library unit
        -- elaboration according to the semantics of section 10.5 of the LRM.

        -- The set of units (to be elaborated when the load image is
```

```
-- executed) is calculated as follows:
--    (1) For a main program, include every unit in the main program's
--        closure, stopping at (and not including) units which are (i) in
--        worlds loaded as subsystems or (ii) other main programs.
--    (2) For a subsystem, for each "selected" library unit in the
--        subsystem, include its unit closure, stopping at (and not
--        including) units which are (i) in other worlds loaded as
--        subsystems or (ii) main programs. The "selected" units could
--        be specified in a variety of ways; examples include: (a)
--        every unit in the world view, (b) every unit in some export
--        list, (c) units specified by some wildcarded pathname; how
--        they are specified is immaterial to this package.
-- Note that, in both cases, this set of units may include units that
-- are in worlds other than the world containing the loaded main
-- program or subsystem.

-- The world reconstruction algorithms allow load images to reference
-- only those code segments which live in the same world as the load
-- image. The purpose of the primary/secondary mechanism is to allow
-- the loading of a subsystem or main program across world boundaries,
-- in spite of this restriction.

-- Load images contain the following information:

--  (1) Ignoring the primary/secondary issues, a load image references
--      the set of code segments which contain the code for the set
--      of units elaborated by executing the load image.

--      Code segment names are NOT stored in code segments themselves,
--      for the following reasons:

--      First, code segment names are local to the machine on which
--      the segment was created. Therefore, keeping code segment names
--      in just the load image object simplifies the process of moving
--      a loaded program/subsystem between machines.

--      Second, code segment names are treated as protected objects by
--      OM (for access control reasons); this would be considerably
--      more difficult if code segment names could be stored in code
--      segments (which would imply the ability to create them from
--      integers).

--      Third, expunge control for code segments follows these rules:
--      A code segment cannot be expunged if there is a load image
--      which references the code segment. A load image cannot be
--      expunged if it is Open. The execution mechanisms keep the
--      load image open as long as there is an elaborated copy of the
--      load image. And the execution mechanisms prevent unelaboration
--      while importers still exist. This acheives the invariant that
--      a code segment cannot be expunged while being executed.
--      Violating this invariant can cause executing programs to get
--      strange exceptions (like Non_Existant_Page_Error) and risks
--      crashing the machine.

--  (2) The load image stores a map from original Object_Index to
--      the Version_Handle of the corresponding Diana tree. Debug
--      tables use these object indexes to refer to units. Use of
--      Version_Handle allows tools to use the universe component
```

```
--       to record the identity of the view from which the unit
--       was loaded. An option is available to expunge protect the
--       referenced trees.

--   (3) For subsystems, the load image stores export Compatability_Keys
--       for all of the units exported by the subsystem.

--   (4) For subsystems, the load image identifies the set of exported
--       runtime modules (corresponding in a perhaps package intergrated
--       fashion to the exported units).

--   (5) The load image identifies the set of subsystems which are
--       imported by the given main program or subsystem. And for each
--       imported subsystem, stores the import Compatability_Keys for
--       the imported units. At execution time, these import keys are
--       matched against the exporter's export keys.

--   (6) The importing load image specifies where (in its import space)
--       it wants to find the module names of the imported runtime
--       modules.

--   (7) The load image identifies the set of main programs which it
--       may call.

--   (8) The elaboration order (chosen by the loader) is recorded in the
--       load image.  This may be useful information for the user.  But
--       the major purpose is so the subsystem integrator (discussed
--       further below) can produce a single output load image from many
--       input load images, and preserve elaboration order.

-- In the event that a loaded main program or subsystem needs to
-- elaborate units in more than one world, the primary/secondary
-- mechanism is used.  A primary load image is created in the world
-- containing the main program or subsystem.  Secondary load images
-- are created in the other worlds.  The primary references all of
-- the secondaries.  The secondaries name the primary.  As long as
-- the primary continues to exist, the secondaries will not (by
-- default) be expunged.  Each load image references the loaded
-- code segments in the same world.

-- Functions in this package return garbage (typically 0) when given
-- garbage parameters.  Procedures which do not return status are
-- noops when given garbage parameters.

-- The functions require the handle to be open for either Read_Object
-- or Write_Object.  The modifying procedures require the handle to
-- be open for Write_Object.

procedure Define_Shape (The_Load_Image : in out Object.Open_Handle;
                        Is_Primary : Boolean;
                        Code_Segment_Count : Natural;
                        Secondary_Count : Natural;
                        Exported_Unit_Count : Natural;
                        Exported_Module_Count : Natural;
                        Imported_Subsystem_Count : Natural;
                        Imported_Unit_Count : Natural;
                        Imported_Program_Count : Natural;
                        Debug_Unit_Count : Natural;
```

```
                                Status : out Error.Condition);

        -- This operation must be the first opreratino to be applied to a
        -- newly created version of a load image. Other operations are used
        -- to "fill in the shape".

        -- When Is_Primary is true, Number_Of_Secondaries indicates the
        -- exact number of secondary load images which will be referenced by
        -- this primary.  Recall that secondary load images are only used when
        -- the load closure spans worlds.

        -- Note that the operations below which set string values will
        -- generate garbage (in the load image) if the string value is set
        -- more than once.  The garbage cannot be reclaimed without expunging
        -- the version.

        -- Possible errors:
        --   Is_Execution_Error (Status)
        --         o Not_A_Load_Image
        --         o Load_Image_Is_Too_Big
        --         o Exported_Module_Count_Out_Of_Bounds

      procedure Set_Code_Segment
                      (The_Load_Image : in out Object.Open_Handle;
                       Key : Code_Segment_Key;
                       Code_Version : Om.Version_Handle;
                       Status : out Error.Condition);

      -- The Object.Code_Name function is used (on Code_Version) to set both
      -- the Current_Segment and Original_Segment attributes. The
      -- Code_Version attribute is assigned the value of the Code_Version
      -- parameter.

      -- Possible errors:
      --   Is_Execution_Error (Status)
      --         o Not_A_Load_Image
      --         o Code_Segment_Key_Out_Of_Bounds
      --         o Not_A_Code_Version
      --               The Code_Version parameter does not refer to an existing
      --               version of an object of Class_Code.
      --         o Must_Be_In_Load_Image_World
      --               Code_Version must be in the same world as the load
      --               image.

      procedure Set_Original_Segment
                      (The_Load_Image : in out Object.Open_Handle;
                       Key : Code_Segment_Key;
                       Original_Segment : Machine.Module_Name);

      -- Used by tool to move a load image between machines.

      -- The Original_Segment and Current_Segment attributes might be
      -- different (for a given Key value) because the load image and the
      -- corresponding code segments were moved from the producing
      -- (original) machine to the current machine.  Retaining
      -- Original_Segments may be useful for debugging.

       -- Possible errors:
```

```
        --    Is_Execution_Error (Status)
        --         o Not_A_Load_Image
        --         o Code_Segment_Key_Out_Of_Bounds

        function Original_Segment (The_Load_Image : Object.Open_Handle;
                                   Key : Code_Segment_Key)
                    return Machine.Module_Name;
        function Current_Segment (The_Load_Image : Object.Open_Handle;
                                  Key : Code_Segment_Key)
                    return Machine.Module_Name;
        function Code_Version (The_Load_Image : Object.Open_Handle;
                               Key : Code_Segment_Key)
                    return Om.Version_Id;

        -- The above functions are used to return the values of the attributes
        -- set by the Set_Code_Segment operation. Note that there is no
        -- visible operation to retrieve the "protected form" of the code
        -- segment name. The elaboration operations use internal facilities
        -- which return the protected form and may generate access control
        -- error conditions.

        function Protect_Debug_Units (The_Load_Image : Object.Open_Handle)
                    return Boolean;
        procedure Set_Protect_Debug_Units
                    (The_Load_Image : in out Object.Open_Handle;
                     Value : Boolean;
                     Status : out Error.Condition);

        -- Defaults to off. When turned on, the references to Diana trees
        -- (supplied by the operation below) will be expunge protected.

        -- Possible errors:
        --  Is_Execution_Error (Status)
        --       o Not_A_Load_Image
        --       o Cant_Protect_Debug_Units
        --            This operation cannot be applied once a call to
        --            Set_Debug_Unit has been made.

        function Debug_Unit (The_Load_Image : Object.Open_Handle;
                             Original_Index : Om.Object_Index)
                    return Om.Version_Handle;
        procedure Set_Debug_Unit (The_Load_Image : in out Object.Open_Handle;
                                  Original_Index : Om.Object_Index;
                                  Debug_Unit : Om.Version_Handle;
                                  Status : out Error.Condition);

        -- Map from original Object_Index to the Version_Handle of the
        -- corresponding Diana tree. Debug tables use these object indexes to
        -- refer to units.  Use of Version_Handle allows tools to use the
        -- universe component to record the identity of the view from which
        -- the unit was loaded.

        -- Possible errors:
        --  Is_Execution_Error (Status)
        --       o Not_A_Load_Image
        --       o Debug_Unit_Index_Out_Of_Bounds
        --       o Not_A_Debug_Unit
        --            The referenced Debug_Unit does not refer to an
```

```
--              existing object of Class_Ada. Only occurs when
--              Protect_Debug_Units is turned on.

-------------------------------------------------------------------
-- The following operations only apply to secondary load images --
-------------------------------------------------------------------

function Protecting_Primary
              (The_Load_Image : Object.Open_Handle)
              return Om.Version_Handle;

procedure Set_Protecting_Primary
               (The_Load_Image : in out Object.Open_Handle;
                Protecting_Primary : Om.Version_Handle;
                Status : out Error.Condition);

-- Protecting_Primary refers to a primary load image which will
-- eventually reference this secondary. Recall that such a reference
-- protects the secondary from expunge in the secondary's world.

-- Possible errors:
--   Is_Execution_Error (Status)
--        o Not_A_Load_Image
--        o Not_A_Secondary_Load_Image
--        o Primary_Load_Image_Does_Not_Exist

-------------------------------------------------------------------
-- The following operations only apply to primary load images --
-------------------------------------------------------------------

function Secondary_Count (The_Load_Image : Object.Open_Handle)
              return Natural;

function Secondary_Reference
              (The_Load_Image : Object.Open_Handle;
               Key : Secondary_Number)
              return Om.Version_Handle;

procedure Set_Secondary_Reference
               (The_Load_Image : in out Object.Open_Handle;
                Key : Secondary_Number;
                Secondary : Om.Version_Handle;
                Status : out Error.Condition);

-- Used to record, in a primary, the Version_Id of one of its
-- secondaries.

-- Possible errors:
--   Is_Execution_Error (Status)
--        o Not_A_Load_Image
--        o Not_A_Primary_Load_Image
--        o Secondary_Load_Image_Does_Not_Exist

function Starting_Pc
              (The_Load_Image : Object.Open_Handle)
              return Machine.Address;

procedure Set_Starting_Pc
```

```
                    (The_Load_Image : in out Object.Open_Handle;
                     Starting_Pc : Machine.Address;
                     Status : out Error.Condition);

   -- Used to remember the starting pc.

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --         o Not_A_Load_Image
   --         o Not_A_Primary_Load_Image
   --         o No_Code_Segment_For_Pc

   function Is_Subsystem
                   (The_Load_Image : Object.Open_Handle) return Boolean;
   procedure Set_Is_Subsystem
                    (The_Load_Image : in out Object.Open_Handle;
                     Value : Boolean;
                     Status : out Error.Condition);

   -- Defaults to off. Determines whether you get subsystem or main
   -- program behaviour during elaboration.

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --         o Not_A_Load_Image
   --         o Not_A_Primary_Load_Image

   function Is_Persistent
                   (The_Load_Image : Object.Open_Handle) return Boolean;
   procedure Set_Is_Persistent
                    (The_Load_Image : in out Object.Open_Handle;
                     Value : Boolean;
                     Status : out Error.Condition);

   -- Defaults to off. The elaboration of a persistent subsystem will
   -- still exist after the command which caused its elaboration.

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --         o Not_A_Load_Image
   --         o Not_A_Primary_Load_Image
   --         o Not_A_Subsystem_Load_Image

   function Exported_Unit_Count
                   (The_Load_Image : Object.Open_Handle) return Natural;
   function Exported_Module_Count
                   (The_Load_Image : Object.Open_Handle) return Natural;

   function Exported_Unit_Id
                   (The_Load_Image : Object.Open_Handle;
                    Unit : Om.Object_Index)
                   return Om.Object_Id;
   function Exported_Unit_Name
                   (The_Load_Image : Object.Open_Handle;
                    Unit : Om.Object_Index)
                   return String;
   function Exported_Module_Pos
                   (The_Load_Image : Object.Open_Handle;
```

```
                Unit : Om.Object_Index)
               return Exported_Module_Position;
    function Exported_Decls
               (The_Load_Image : Object.Open_Handle;
                Unit : Om.Object_Index)
               return Compatability_Vector;

    procedure Set_Exported_Unit_Information
               (The_Load_Image : in out Object.Open_Handle;
                Unit_Id : Om.Object_Id;
                Unit_Name : String;
                Module_Pos : Exported_Module_Position;
                Exported_Decls : Compatability_Vector;
                Status : out Error.Condition);

    -- The exported unit information is used (for a loaded subsystem) to
    -- describe the actual declarations that are exported (via the
    -- Compatability_Vector for each exported unit). This information is
    -- used for compatability checking during elaboration.

    -- Under library unit package integration, more than one exported unit
    -- may specify the same Exported_Module_Position. During elaboration,
    -- the module will call Operations.Set_Exported_Module with the Module
    -- parameter equal to the value of Module_Pos supplied here.

    -- Possible errors:
    --   Is_Execution_Error (Status)
    --        o Not_A_Load_Image
    --        o Not_A_Primary_Load_Image
    --        o Not_A_Subsystem_Load_Image
    --        o Exported_Module_Pos_Out_Of_Bounds
    --        o Exported_Unit_In_Wrong_World
    --             The Unit_Id of an exported unit must specify the
    --             same World_Number as that of the load image being
    --             constructed.
    --        o Load_Image_Is_Too_Big


    function Imported_Subsystem_Count
               (The_Load_Image : Object.Open_Handle) return Natural;

    function Imported_Subsystem_Id
               (The_Load_Image : Object.Open_Handle;
                Subsystem : Imported_Subsystem_Number)
               return Root_Of_World;
    function Imported_Subsystem_Name
               (The_Load_Image : Object.Open_Handle;
                Subsystem : Imported_Subsystem_Number)
               return String;
    function Imported_Subsystem_First_Unit
               (The_Load_Image : Object.Open_Handle;
                Subsystem : Imported_Subsystem_Number)
               return Imported_Unit_Number;
    function Imported_Subsystem_Last_Unit
               (The_Load_Image : Object.Open_Handle;
                Subsystem : Imported_Subsystem_Number)
               return Imported_Unit_Number;
```

```
   procedure Set_Imported_Subsystem_Information
                  (The_Load_Image : in out Object.Open_Handle;
                   Subsystem : Imported_Subsystem_Number;
                   Subsystem_Id : Some_Object_In_World;
                   Subsystem_Name : String;
                   First_Imported_Unit : Imported_Unit_Number;
                   Last_Imported_Unit : Imported_Unit_Number;
                   Last_Imported_Module : Imported_Unit_Number;
                   Status : out Error.Condition);

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --        o Not_A_Load_Image
   --        o Not_A_Primary_Load_Image
   --        o Not_A_Subsystem_Load_Image
   --        o Imported_Subsystem_Number_Out_Of_Bounds
   --        o Imported_Subsystem_Does_Not_Exist
   --        o Imported_Unit_Number_Out_Of_Bounds
   --        o Load_Image_Is_Too_Big

   function Imported_Unit_Count
                  (The_Load_Image : Object.Open_Handle) return Natural;

   function Imported_Unit_Id
                  (The_Load_Image : Object.Open_Handle;
                   Unit : Imported_Unit_Number)
                  return Om.Object_Handle;
   function Imported_Unit_Name
                  (The_Load_Image : Object.Open_Handle;
                   Unit : Imported_Unit_Number)
                  return String;
   function Imported_Module_Pos
                  (The_Load_Image : Object.Open_Handle;
                   Unit : Imported_Unit_Number)
                  return Imported_Module_Position;
   function Imported_Decls
                  (The_Load_Image : Object.Open_Handle;
                   Unit : Imported_Unit_Number)
                  return Compatability_Vector;

   procedure Set_Imported_Unit_Information
                  (The_Load_Image : in out Object.Open_Handle;
                   Unit : Imported_Unit_Number;
                   Unit_Id : Om.Object_Id;
                   Unit_Name : String;
                   Module_Pos : Imported_Module_Position;
                   Imported_Decls : Compatability_Vector;
                   Status : out Error.Condition);

   -- Both subsystems and main programs can import the facilities of
   -- other subsystems. Since environment facilities are imported using
   -- this mechanism, it is unlikely that any load image has 0
   -- subsystem imports. For each imported subsystem, there is a set of
   -- imported units (as indicated by the First_Imported_Unit and
   -- Last_Imported_Unit values). For each imported unit, one specifies
   -- the declarations (via the Compatability_Vector) which are actually
   -- imported. This information is matched against the unit export
   -- information (in the load image of the corresponding imported
```

```
   -- subsystem) during compatability checking during elaboration.

   -- During elaboration the names of the imported runtime modules appear
   -- in the import space by Imported_Module_Position; this value may be
   -- different from the Exported_Module_Position of the corresponding
   -- unit.

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --        o Not_A_Load_Image
   --        o Not_A_Primary_Load_Image
   --        o Not_A_Subsystem_Load_Image
   --        o Imported_Unit_Number_Out_Of_Bounds
   --        o Imported_Module_Pos_Out_Of_Bounds
   --        o Imported_Unit_Does_Not_Exist
   --        o Load_Image_Is_Too_Big

   function Needs_Ucode_Assist (The_Load_Image : Object.Open_Handle)
                 return Boolean;
   procedure Set_Needs_Ucode_Assist
                  (The_Load_Image : in out Object.Open_Handle;
                   Value : Boolean;
                   Status : out Error.Condition);

   -- Defaults to off. This is a special kludge which is used to cause a
   -- subsystem to import the ucode assist package. It is required by the
   -- "system call and command subsystems". A subsystem with this bit set
   -- can only be elaborated by Privileged users.

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --        o Not_A_Load_Image
   --        o Not_A_Primary_Load_Image
   --        o Not_A_Subsystem_Load_Image


   subtype Load_Image_Id is Om.Version_Handle;

   function Imported_Program_Count
                 (The_Load_Image : Object.Open_Handle) return Natural;

   function Imported_Program_Id
                 (The_Load_Image : Object.Open_Handle;
                  Program : Imported_Program_Number)
                 return Load_Image_Id;
   function Imported_Program_Name
                 (The_Load_Image : Object.Open_Handle;
                  Program : Imported_Program_Number)
                 return String;

   procedure Set_Imported_Program_Information
                  (The_Load_Image : in out Object.Open_Handle;
                   Program : Imported_Program_Number;
                   Program_Id : Load_Image_Id;
                   Program_Name : String;
                   Status : out Error.Condition);

    -- Both subsystems and main programs can call other main programs.
```

```
    -- The identity of these programs is recorded as shown above.

    -- Possible errors:
    --   Is_Execution_Error (Status)
    --         o Not_A_Load_Image
    --         o Not_A_Primary_Load_Image
    --         o Imported_Program_Number_Out_Of_Bounds
    --         o Imported_Program_Does_Not_Exist
    --         o Load_Image_Is_Too_Big


    function Elaboration_Order (The_Load_Image : Object.Open_Handle)
                  return Elaboration_Order_Array;
    procedure Set_Elaboration_Order
                  (The_Load_Image : in out Object.Open_Handle;
                   Order : Elaboration_Order_Array;
                   Status : out Error.Condition);

    -- The elaboration order is stored in the load image so that the
    -- subsystem integrator can produce an identical elaboration order
    -- in the target integrated subsystem.

    -- Possible errors:
    --   Is_Execution_Error (Status)
    --         o Not_A_Load_Image
    --         o Not_A_Primary_Load_Image
    --         o Elaboration_Order_Length_Mismatch
    --               The 'length of the Order parameter must be the same
    --               as the Unit_Count parameter to Define_Shape.


    function Enable_Code_Acl_Check
                  (The_Load_Image : Object.Open_Handle) return Boolean;
    procedure Set_Enable_Code_Acl_Check
                  (The_Load_Image : in out Object.Open_Handle;
                   Value : Boolean;
                   Status : out Error.Condition);

    -- Defaults to off. When enabled, a job must have Execute access to
    -- the load image in order to elaborate a new copy.

    -- Possible errors:
    --   Is_Execution_Error (Status)
    --         o Not_A_Load_Image
    --         o Not_A_Primary_Load_Image


    function Enable_Module_Acl_Check
                  (The_Load_Image : Object.Open_Handle) return boolean;
    procedure Set_Enable_Module_Acl_Check
                  (The_Load_Image : in out Object.Open_Handle;
                   Value : Boolean;
                   Status : out Error.Condition);

    -- Defaults to off. When enabled, a job must have Execute access to
    -- the load image in order to import modules from an already elaborated
    -- persistent copy of the subsystem.
```

```
        -- Possible errors:
        --   Is_Execution_Error (Status)
        --         o Not_A_Load_Image
        --         o Not_A_Primary_Load_Image


        function Original_Machine_Name
                        (The_Load_Image : Object.Open_Handle) return String;
        procedure Set_Original_Machine_Name
                        (The_Load_Image : in out Object.Open_Handle;
                         Name : String);

         -- The originating machine name is useful for error messages and
         -- debugging.

         -- Possible errors:
         --   Is_Execution_Error (Status)
         --         o Not_A_Load_Image
         --         o Not_A_Primary_Load_Image
         --         o Load_Image_Is_Too_Big

        function Loader_version
                        (The_Load_Image : Object.Open_Handle) return String;
        procedure Set_Loader_Version
                        (The_Load_Image : in out Object.Open_Handle;
                         Version : String);

         -- The name and/or version and/or timestamp of the "loader" is useful
         -- for error messages and debugging.

         -- Possible errors:
         --   Is_Execution_Error (Status)
         --         o Not_A_Load_Image
         --         o Not_A_Primary_Load_Image
         --         o Load_Image_Is_Too_Big

        function Program_Version
                        (The_Load_Image : Object.Open_Handle) return String;
        procedure Set_Program_Version
                        (The_Load_Image : in out Object.Open_Handle;
                         Version : String);

         -- The program/subsystem might find this useful for its own
         -- diagnostic purposes. Can contain program name, and/or version,
         -- and/or timestamp.

         -- Possible errors:
         --   Is_Execution_Error (Status)
         --         o Not_A_Load_Image
         --         o Not_A_Primary_Load_Image
         --         o Load_Image_Is_Too_Big

    end Load_Image;
    --pragma Integrate (Load_Image);

    -------------------------
    -- Execution Operations --
    -------------------------
```

```
package Operations is

    ------------------------------------------
    -- Operations to control elaboration state --
    ------------------------------------------

    -- The execution stack follows the same conventions as the various
    -- job stacks, with some exceptions: (a) certain kinds of stack
    -- entries cannot be copied (namely, direct entries); (b) it is
    -- not possible to modify stack entries below top of stack.

    -- Each stack element contains an entity which comes in 2 classes,
    -- known as "direct" and "indirect" execution states. A indirect is
    -- simply a reference to a direct.  This capability can be used to
    -- acheive "shared elaboration". A direct execution state is a map
    -- from persistent subsystem to physically elaborated copy of the
    -- persistent subsystem.

    -- When we say "physically elaborate" we mean that the elaborating
    -- job causes the execution of a new runtime copy of the subsystem.
    -- This causes the top (direct) execution state of the job to get a
    -- map entry which selects the new copy; this kind of map entry is
    -- known as a local entry.

    -- It is also possible to cause a direct execution state to contain
    -- a map entry which selects a copy which is physically elaborated
    -- by another job. This capability can be used to acheive "shared
    -- elaboration". This kind of map entry is known as a non-local entry.

    -- When a direct execution state contains a map entry for a subsystem
    -- S, the subsystem S is said to be "elaborated" in the given
    -- execution state (regardless of whether or not the entry is local).

    -- Each direct execution state is "consistent". In particular, the
    -- following invariants hold (within the context of the particular
    -- execution state):
    --     (1) Consider subsystem A which is elaborated.  Consider a
    --         subsystem X which is with'd by A.
    --            (a) X is also elaborated.
    --            (b) The map entry for X references (directly or
    --                indirectly) the physically elaborated copy of X
    --                which is imported by A.
    --            (c) The import Compatability_Vector's of A "match" the
    --                corresponding export Compatability_Vector's of the
    --                physically elaborated copy of subsystem X which is
    --                imported by A.
    --     (2) By induction, one can infer that if 2 elaborated subsystems
    --         A and B with X, they import from the same physically
    --         elaborated copy of X.

    -- Indirect execution state are by definition consistent, since they
    -- simply reference a consistent direct execution state.

    -- "Sharing" is acheived as follows: Job X elaborates some subsystems,
    -- and invokes a Make_Execution_State_Shareable operation on the
    -- particular execution state. Job X may optionally provide a string
    -- name to identify the execution state. Job Y (the "importer") can
```

```
    -- push an indirect execution state by invoking the
    -- Share_Execution_State operation, supplying the string name, or the
    -- explicit job and stack location. At this point, Y can run programs,
    -- having them import from the shared subsystems (which Y imported
    -- from X). If Y wishes to physically elaborate its own subsystems, it
    -- can Push_Direct_State. Elaboration operations in job Y can then
    -- create local map entries for Y's private subsystems, and can
    -- create non-local map entries from the indirect execution state (at
    -- Top-1). This may become clearer after reading further.

    -- In order for persistently elaborated subsystems to last longer than
    -- the command which caused their elaboration, we introduce the
    -- concept of a "subsystem state" job in which persistently elaborated
    -- subsystems live.  There is at most one of these jobs per session.

    -- We will use the term "elaborated" to mean that an elaborated copy
    -- of the subsystem is selected by the execution state on top of the
    -- stack. The copy might be selected via an indirect execution state,
    -- or via a direct state with either a local or non-local map entry.
    -- Some consequences of this definition: (a) Pushing an empty direct
    -- state results in no subsystem being "elaborated".  (b) A subsystem
    -- can be said to be "not elaborated", even though lower stack
    -- elements reference physically elaborated copies.  (c) Thus, the
    -- term "elaborated" is considerably different than the term
    -- "physically elaborated".


    type Elaboration_Options is new Long_Integer;
    Create_Job_On_Demand         : constant Elaboration_Options := 2 ** 0;
    Push_Direct_State_On_Demand  : constant Elaboration_Options := 2 ** 1;
    Force_Private_Copy           : constant Elaboration_Options := 2 ** 2;
    Allow_Private_Copy           : constant Elaboration_Options := 2 ** 3;
    Elaborate_Closure_On_Demand  : constant Elaboration_Options := 2 ** 4;
    Ignore_Compatability_Errors  : constant Elaboration_Options := 2 ** 5;

    Default_Elab_Options         : constant Elaboration_Options :=
                                   Create_Job_On_Demand +
                                   Push_Direct_State_On_Demand +
                                   Allow_Private_Copy +
                                   Elaborate_Closure_On_Demand;


    generic
        with procedure Visit (Subsystem : Root_Of_World;
                              Name : String;
                              Elaborated_Release : Om.Version_Handle;
                              Physically_Elaborated : Boolean;
                              Where_Elaborated : Om.Job_Id);

    procedure Elaborate_Persistent_Subsystem
         (Subsystem : Some_Object_In_World;
          Status : out Error.Condition;
          Exception_Info : out Exception_Information;
          Options : Elaboration_Options := Default_Elab_Options;
          In_Job : Om.Job_Id := Job.Subsystem_State);

    -- The purpose of this operation is to elaborate the subsystem, if
    -- its not already elaborated.
```

```
   -- If In_Job is Nil, and Create_Job_On_Demand is enabled, the
   -- operation will create a Subsystem_State job for the session of the
   -- calling job.

   -- If Push_Direct_State_On_Demand is enabled, the operation will do
   -- an implicit call to Push_Direct_State, if necessary.

   -- If Force_Private_Copy is enabled, then this operation will
   -- physically elaborate a new copy of every subsystem (in the given
   -- Subsystem's with closure, as well as Subsystem itself), in the
   -- identified subsystem job, for every subsystem which is not
   -- already elaborated.

   -- If Allow_Private_Copy is not enabled, then this operation is limited
   -- to creating map entries to already physically elaborated copies of
   -- subsystems.

   -- If Elaborate_Closure_On_Demand is enabled, then this operation
   -- will, if necessary, attempt to elaborate "lower level" subsystems
   -- (ie, subsystems in the "with" closure).

   -- This operation applies to the execution state stack of the
   -- identified job.

   -- In the course of elaborating the needed subsystems, consider the
   -- situation in which the with closure of persistent subsystem S is
   -- elaborated, but S itself is not yet elaborated (and is the current
   -- goal).  Note that if stack top is not direct and
   -- Push_Direct_State_On_Demand is enabled, an implicit call to
   -- Push_Direct_State will be made. The following actions are taken to
   -- elaborate S:

   -- If Force_Private_Copy is disabled, then look for a shareable copy
   -- by scanning the execution stack stopping at the first element which
   -- contains a map entry for S. Note that when this scan encounters
   -- indirect execution states, it simply looks at the referenced
   -- execution state. If a map entry is found, and the referenced
   -- physically elaborated copy of S imports from the same physically
   -- elaborated copies that are currently selected by the top execution
   -- state, then simply create a non-local map entry (in stack top)
   -- referencing the located copy. Calls the supplied Visit procedure
   -- prior to setting the map entry; the call would be made with
   -- Physically_Elaborated => False.

   -- If Force_Private_Copy is enabled, or the search for a shareable
   -- copy failed, then attempt to physically elaborate a new copy of
   -- subsystem S, importing from the copies elaborated in stack top,
   -- creating a local map entry in stack top.  This step may fail with
   -- Runtime_Compatability_Mismatch.  Calls the supplied Visit procedure
   -- prior to physical elaboration; the call would be made with
   -- Physically_Elaborated => True.  Note that enabling both
   -- Force_Private_Copy and Include_Closure can result in the physical
   -- elaboration of many subsystems.

   -- When physically elaborating a new copy, the load image is
   -- identified by the world view specified by the job's view stack.
```

```
   -- when physically elaborating a new copy, the runtime modules
   -- execute in the identified job.

   -- The caller suspends until the physically elaborated copies have
   -- finished elaboration of their library units. Clearly, the runtime
   -- state of these units survives the completion of this operation.

   -- In the event of an error, the offending subsystem is unelaborated,
   -- but the other subsystems that were elaborated are left elaborated.

   -- There are notes at the end of this package describing the context
   -- in which the code (in the load image) begins execution.

   -- Note that since Visit is called prior to elaboration, many of the
   -- error conditions are known to apply to the subsystem last Visit'd.

   -- The Name parameter to Visit comes from the Program_Version attribute
   -- of the elaborated load image.

   -- Possible errors:
   --   Is_Job_Error (Status)
   --        o Job_Does_Not_Exist
   --             Non-Nil In_Job parameter refers to non-existant job.
   --   Is_Execution_Error (Status)
   --        o Missing_Subsystem_Job
   --             Can only happen when Create_Job_On_Demand is disabled.
   --        o Not_A_Subsystem_Job
   --             The identified job must be the subsystem job for
   --             some session.
   --        o Execution_Stack_Is_Empty
   --             Can only happen when Push_Direct_State_On_Demand
   --             is disabled.
   --        o Execution_State_Not_Direct
   --             Can only happen when Push_Direct_State_On_Demand
   --             is disabled.
   --        o Subsystem_Is_Already_Elaborated
   --             The identified subsystem is already elaborated. In some
   --             cases caller might consider this to be "success".
   --        o Private_Copy_Not_Allowed
   --             Couldn't find an import compatable already physically
   --             elaborate copy of a subsystem, and Allow_Private_Copy is
   --             disabled.
   --        o Cant_Find_Load_Image_To_Elaborate
   --             Either the job's universe stack does not select a
   --             world view for the subsystem, or the world view does
   --             not identify an existing load image.
   --        o Cycles_In_Subsystem_Closure
   --             Subsystem with dependencies induce a cylic order.
   --        o Cycles_in_View_Indirection
   --             Cycle detected determining the target universe view.
   --        o Subsystem_Not_Persistent
   --             The identified Subsystem is not persistent.
   --        o Persistent_Withs_Non_Persistent
   --             A subsystem (in the closure) withs a non-persistent
   --             subsystem.
   --        o Runtime_Compatability_Mismatch
   --             Subsystem A needs to be elaborated, and it with's
   --             subsystem B. A imports units and/or declarations which
```

```
--              are not exported by B. This is detected by the fact
--              that an import Compatability_Vector in the load image
--              for A does not "match" the corresponding export
--              Compatability_Vector in the load image for B.
--          o Broken_Indirect_Execution_State
--              The search for an elaborated copy ran across an
--              indirect execution state (in the job's stack) which
--              has been broken (perhaps by termination of the
--              referenced job).
--          o Subsystem_Propagated_Exception
--              The elaboration of a library unit propagated an
--              exception. Note that it is a consequence of Ada
--              semantics that such exceptions cannot be propagated
--              once this operation returns. The Exception_Info
--              parameter contains additional information.
--    Is_Access_Control_Error (Status)
--          o Couldn't load code segments or import modules or import
--            ucode assist due to an access control violation. See
--            Load_Image packages for details. Or don't have required
--            access to the identified job.


   procedure Make_Execution_State_Shareable
                  (Status : out Error.Condition;
                   Name : String := "";
                   In_Job : Om.Job_Id := Job.Subsystem_State);

   -- The top stack location (of the given job) is marked as shareable.
   -- If the Name parameter is not null, then the name attribute of the
   -- top stack location is set to the given Name. If the resulting top
   -- stack location ends up with a non-null Name attribute, invocations
   -- of Shared_Execution_State will able to identify this state using
   -- its string name. Names longer than about 128 characters are
   -- truncated on the left.

   -- Note that this job will not be able to terminate (or be Job.Kill'd)
   -- if other job's are currently sharing this job's subsystems. Some of
   -- the motivation for this invariant is "politeness"; but more
   -- importantly, the machine might crash if the rule were violated.

   -- Possible errors:
   --    Is_Job_Error (Status)
   --          o Job_Does_Not_Exist
   --    Is_Execution_Error (Status)
   --          o Not_A_Subsystem_Job
   --          o Execution_Stack_Is_Empty
   --          o Execution_State_Not_Direct
   --          o Execution_State_Is_Shareable
   --          o Execution_State_Name_Not_Unique
   --              When non-null, the string name of the execution state
   --              must be unique among all of the execution states (in the
   --              entire system) which are marked as shareable.

   procedure Make_Execution_State_Not_Shareable
                  (Status : out Error.Condition;
                   In_Job : Om.Job_Id := Job.Subsystem_State);

   -- The top stack location (of the identified job) is made not shareable.
```

```
        -- Will break any referencing indirect execution states. Note that
        -- this operation does NOT unelaborate anything. And it does NOT
        -- break non-local map entries (of direct execution states) that have
        -- been constructed by elaboration via previously unbroken indirect
        -- execution states.

        -- Possible errors:
        --   Is_Job_Error (Status)
        --        o Job_Does_Not_Exist
        --   Is_Execution_Error (Status)
        --        o Not_A_Subsystem_Job
        --        o Execution_Stack_Is_Empty
        --        o Execution_State_Not_Direct
        --        o Execution_State_Not_Shareable


        procedure Share_Execution_State
                        (Source_State : String;
                         Status : out Error.Condition;
                         Target_Stack : Job.Stack_Id := Job.Job_Stack);
        procedure Share_Execution_State
                        (Source_State : Om.Job_Id;
                         Status : out Error.Condition;
                         Source_Job : Om.Job_Id := Job.Subsystem_State;
                         Source_Location : Job.Stack_Location := Job.Top;
                         Target_Stack : Job.Stack_Id := Job.Job_Stack);

        -- The first form identifies the source execution state by string
        -- name, whereas the second form identifies the source execution state
        -- by explicitly giving the job and stack location.  Clearly, the
        -- presence of concurrency may make the second form subject to
        -- misinterpretation of stack position.

        -- In either case, an indirect elaboration state is pushed on the
        -- identified target stack.

        -- Invocation of Make_Execution_State_Unshareable (on the referenced
        -- elaboration state) will cause the pushed indirect elaboration state
        -- to become broken and cause Elaborate_Persistent_Subsystem to
        -- generate errors. This may happen automatically when a job
        -- terminates.

        -- Recall that this is what allows the Elaborate_Persistent_Subsystem
        -- operation to elaborate non-local map entries and thereby allows one
        -- job to physically elaborate subsystems against subsystems
        -- physically elaborated by another job. And it allows the
        -- Elaborate_Program operation to directly import from the subsystem
        -- physically elaborated by another job.

        -- Possible errors:
        --   Is_Job_Error (Status):
        --        o Job_Does_Not_Exist
        --             Applies to either source or target job.
        --   Is_Execution_Error (Status):
        --        o Execution_State_Does_Not_Exist
        --             The given string (or given job & stack location) do
        --             not identify a execution state.
        --        o Execution_Stack_Is_Empty
```

```
--              Applies to source job.
--          o Execution_State_Not_Direct
--              Applies to source job.
--          o Execution_State_Not_Shareable
--              Applies to source job. The identified execution state
--              has not been the subject of
--              Make_Execution_State_Shareable operation.
--          o Execution_State_Stack_Overflow
--              Applies to target job.


procedure Un_Elaborate_Persistent_Subsystem
                (Subsystem : Some_Object_In_World;
                 Status : out Error.Condition;
                 Un_Elaborate_Clients : Boolean := True;
                 In_Job : Om.Job_Id := Job.Subsystem_State);

-- Used to unelaborate a persistent subsystem. When
-- Un_Elaborate_Clients is enabled, will unelaborates all the with'ing
-- subsystems as well. Unelaborate means: Deselect the subsystem and
-- physically unelaborate those subsystems which this job physically
-- elaborated. Can use this to kill a partial elaboration. Can be
-- invoked by the given subsystem itself, or one whose closure
-- includes the given subsystem.

-- Possible errors:
--   Is_Job_Error (Status):
--          o Job_Does_Not_Exist
--              Applies to either source or target job.
--   Is_Execution_Error (Status)
--          o Execution_Stack_Is_Empty
--          o Execution_State_Is_Not_Direct
--          o Execution_State_Is_Shareable
--              The identified execution state has been made shareable
--              via the Share_Execution_State operation.
--          o Subsystem_Is_Not_Elaborated
--          o Subsystem_Still_Has_Clients
--              The identified subsystem has clients, but
--              Un_Elaborate_Clients is not enabled.  Note that these
--              clients are in the same execution state.

procedure Elaborate_Program
                (Program_Load_Image : Om.Version_Id;
                 Status : out Error.Condition;
                 Exception_Info : out Exception_Information;
                 Options : Elaboration_Options := Default_Elab_Options;
                 In_Job : Om.Job_Id := Job.Subsystem_State);

-- The purpose of this operation is to execute (ie, elaborate) the
-- identified program.

-- If the Program_Load_Image references a persistent subsystem that is
-- not already elaborated and Elaborate_Closure_On_Demand is enabled,
-- then this operation will invoke Elaborate_Persistent_Subsystem
-- (with the given Options and Job_Id) to get the subsystem elaborated.

-- Elaborates copies (private to this program) of all the
-- non-persistent subsystems in the program's subsystem closure. Runs
```

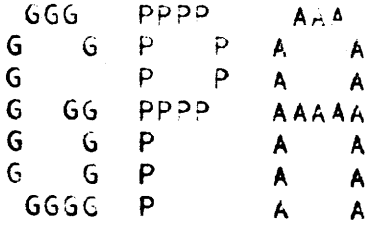
```
   -- the program. When the program returns, unelaborates the
   -- non-persistent copies, but not the persistent copies.

   -- The caller is suspended during the program's execution.

   -- If the task executing the program is aborted, the non-persistent
   -- copies are automatically unelaborated.

   -- Possible errors:
   --   Is_Execution_Error (Status)
   --        o Subsystem_Is_Not_Elaborated
   --            A with'd persistent subsystem is not elaborated, and
   --            Elaborate_Closure_On_Demand is not enabled.
   --        o Plus all those that can be produced by
   --          Elaborate_Persistent_Subsystem


   procedure Create_Subsystem_State_Job
                (Status : out Error.Condition;
                 For_Session : Om.Session_Id := Job.Session);

   -- Can be used to explicitly create the subsystem state job for a
   -- session. Can be done implicitly by the elaboration operations.

   -- Possible errors:
   --   Is_Job_Error (Status):
   --        o Session_Does_Not_Exist
   --   all errors produced by Job.Initiate.

   procedure Push_Direct_State (Status : out Error.Condition;
                                Name : String := "";
                                In_Job : Om.Job_Id := Job.Subsystem_State);

   -- Pushes a fresh elaboration context in which nothing is elaborated.

   -- Possible errors:
   --   Is_Job_Error (Status):
   --        o Job_Does_Not_Exist
   --   Is_Execution_Error (Status):
   --        o Execution_State_Stack_Overflow

   procedure Pop_State (Status : out Error.Condition;
                        Stack : Job.Stack_Id := Job.Job_Stack;
                        Location : Job.Stack_Location := Job.Top);

   -- Will automatically invoke Un_Elaborate_Persistent_Subsystem, if
   -- necessary. Can be applied to shareable execution state, provided
   -- there are no current importers. This operation is used by job
   -- termination (including Job.Kill), and may therefore be responsable
   -- for the inability to kill a job.

   -- Possible errors:
   --   Is_Job_Error (Status):
   --        o Job_Does_Not_Exist
   --   Is_Execution_Error (Status)
   --        o Execution_Stack_Is_Empty
   --        o Subsystem_Still_Referenced
   --            The identified execution state is direct and contains
```

```
--              a local entry for a physically elaborated copy which
--              is still referenced by the direct execution state of
--              some other job.

--------------------------
-- Information requests --
--------------------------

function Stack_Depth (Stack : Job.Stack_Id := Job.Job_Stack)
              return Natural;
function State_Exists (Stack : Job.Stack_Id := Job.Job_Stack;
                       Location : Job.Stack_Location := Job.Top)
              return Boolean;

function Is_Persistently_Elaborated
              (Subsystem : Some_Object_In_World;
               In_Job : Om.Job_Id := Job.Current) return Boolean;

-- Returns true iff the identified job selects an persistent elaborated
-- copy of the subsystem.

function Executing_Release
              (Subsystem : Some_Object_In_World;
               In_Job : Om.Job_Id := Job.Current)
              return String;

-- Simply returns Nil if the identified job has no persistent copy
-- selected and its view stack does not select an executable release
-- of the subsystem. Else, returns the Program_Version attribute of
-- the load image of the job's selected persistent copy or of the load
-- image that would be elaborated (from the view stack).


generic
    with procedure Visit (Job : Om.Job_Id;
                          Load_Image : Om.Version_Handle;
                          Is_Subsystem : Boolean;
                          Name : String);

procedure Locate_Un_Elaborate_Preventing_References
     (Subsystem : Some_Object_In_World;
      In_Job : Om.Job_Id;
      Status : out Error.Condition);

-- will simply produce an error condition if the identified subsystem
-- is not elaborated. Else calls Visit for every subsystem/program
-- which must terminate before the identified subsystem can be
-- unelaborated. The Name string comes from the Program_Version
-- attribute of the visited load image. In some cases, the only valid
-- information will be the Job_Id; in this case, the Load_Image will
-- be Nil.

-- Possible errors:
--   Is_Execution_Error (Status)
--         o Subsystem_Is_Not_Elaborated

generic
    with procedure Visit (Subsystem : Root_Of_World;
```

```
                                Name : String);

    procedure Locate_Job_Termination_Preventing_Subsystems
         (In_Job : Om.Job_Id;
          Status : out Error.Condition);

    -- Will simply produce an error condition if the identified job does
    -- not exist. Else calls Visit for every persistent subsystem
    -- physically elaborated by the identified job and currently shared
    -- with other jobs and which therefore prevent the given job from
    -- terminating. The Name string comes from the Program_Version
    -- attribute of the corresponding load image.

    -- Possible errors:
    --   Is_Execution_Error (Status)
    --         o Subsystem_Is_Not_Elaborated

    ------------------------------------------------------
    -- Operations used by the executing program/subsystem --
    ------------------------------------------------------

    type Elaboration_Handle is private;

    procedure Set_Exported_Module (Subsystem : Elaboration_Handle;
                                   Module : Exported_Module_Position;
                                   Status : out Error.Condition);

     -- This operation is intended for use by an elaborating persistent
     -- subsystem. See code generation notes below.

     -- This operation is a noop if the subsystem is not persistent. It
     -- is used to record the name of an exported runtime module, for
     -- later importation by some subsystem/program.

     -- Possible errors:
     --   Is_Execution_Error (Status)
     --         o Bogus_Elaboration_Handle
     --         o Exported_Module_Pos_Out_Of_Bounds

    type Load_Image_Handle is private;

    -- Handle on the load image object of the running program.

    procedure Elaborate_Program (Program_Load_Image : Load_Image_Handle;
                                 Program : Imported_Program_Number;
                                 Number_Of_Arguments : Natural;
                                 Returns_Result : Boolean;
                                 Status : out Error.Condition);

     -- Logically equivalent to the above form. Intended for use only by
     -- "linkage" code generated by the code generator/loader. The call
     -- site is passing Number_Of_Arguments and expects to receive a
     -- function result if Returns_Result is true. See code generation
     -- notes below.

     -- Possible errors:
     --   all those that can be produced by the previous form of the
     --         operation.
```

```
--    Is_Execution_Error (Status)
--         o Bogus_Load_Image_Handle
--         o Program_Number_Out_Of_Bounds

-------------------------------------
-- Load image execution environment --
-------------------------------------

-- With respect to code generation, the following information is
-- important (if you're not a code generator writer, you probably
-- don't care).

-- The context for executing a subsystem load image is as follows: A
-- skin package (the root module) is elaborated whose import space
-- has things pushed in the following order:
--    o Elaboration_Handle (argument to pass to following op)
--    o Proc ref for Operations.Set_Exported_Module
--    o Load_Image_Handle (argument to pass to following op)
--    o Proc ref for Operations.Elaborate_Program (2nd form)
--    o Code segment names are pushed, in their "protected form", in
--      the following order: for each load image (starting with the
--      primary, and proceeding to the secondaries, in
--      Secondary_Number order) push the code segment names in
--      Code_Segment_Key order.
--    o The imported package variables are pushed in the following
--      order: First push ucode assist (or Nil if not requested).
--      Then, for each imported subsystem (in
--      Imported_Subsystem_Number order) push the package variable
--      for the root module (or nil if the subsystem was not
--      elaborated by this job) followed by the package variables
--      for each imported module (in Imported_Module_Position order).
-- The declaration list of the skin package (root module) runs the
-- code specified by the load image's start pc. This mechanism
-- assumes that the generated code is such that the skin package will
-- complete its elaboration as soon as all the library units (in the
-- subsystem) have completed their elaboration.

-- Note that the skin package is dependent upon some system package,
-- and not the caller's frame. The module name of the skin package
-- (root module) is recorded for later use by Un_Elaborate_Subsystem.

-- The context for executing a program load image is as follows: A
-- subprogram variable is created (using the load image's start pc).
-- The subprogram is called with arguments pushed in the following
-- order:
--    o "Ada" arguments (Number_Of_Arguments)
--    o Load_Image_Handle
--    o Proc ref for Execution.Elaborate_Program (2nd form)
--    o Code segment names (in same order as for subsystems)
--    o Imported package variables (in same order as for subsystems)
-- It is expected to consume all of its arguments, returning 0 or 1
-- results (as specified by the Returns_Results parameter). A
-- result returned from a program executed via the first form of
-- Elaborate_Program is simply ignored. For command execution, it
-- is expected that the job's root thread will be used to execute
-- the program.

------------------------
```

```
        -- Debugger operations --
        -------------------------

        function To_Version_Handle (Program_Load_Image : Load_Image_Handle)
                    return Om.Version_Handle;

        -- Can be used by the debugger to convert the handle (in the
        -- "imports" for the subsystem or program) to a Version_Handle.

         --------------------------------
         -- Observations on Debugging --
         --------------------------------

         -- How does one associate an executing program with a debugger, and
         -- how does its debug state (namely, interface subprogram and break
         -- mask) get set properly? By default, the debug state is nil. If you
         -- tell the code generator the right thing, it will import the
         -- appropriate debugger, and generate code to call it to establish
         -- debugging; it calls the debugger using the normal importing rules
         -- (ie, not with fixed module names).

         -- Suppose one has elaborated a load image, and now one wants to
         -- debug it, which implies finding the Diana trees associated with
         -- the various code segments in the load image. This is an
         -- excellent question. How does it work?

         --------------------
         -- Amplification --
         --------------------

         -- One can get amplification in shared subsystems by doing the
         -- initial elaboration in a job with the appropriate rights.
         -- Currently, there are no mechanisms for automatic amplification.

    private
        type Load_Image_Handle is new Load_Image_Layout.State_Record_Pointer;
        type Elaboration_Handle is
            new Execution_Layout.Subsystem_State_Pointer;

        -- Both types must be "by value" types so that we can stick them
        -- in import spaces without creating dangling references.

    end Operations;
    --pragma Integrate (Operations);

end Execution;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.OM.OM_SERVICES_1
Version: V(18)
Request: 1275

Date: April 24, 1986
Queued: 11:27:01 AM
Printed: 11:41:31 AM

```
with Job;
with Disk;
with Error;
with Naming;
with Object;
with Machine;
with Calendar;
with Om_Definitions;

package Om_Services_1 is

    package Link is
        -- Pragma Subsystem (Object_Management);

        -- A "link pack" is an object that defines a map from simple Ada
        -- names to Ada library units. A "link" is one element of this map.
        -- Each world has exactly one link pack object. Eech world view
        -- potentially has its own version of the link pack.

        -- A link is Internal if its designated object is in the world of the
        -- link pack; otherwise it is External.

        -- As defined by the Object package, there is an internal link for
        -- every Ada library unit; its link name is the same as the simple
        -- name of the designated object. External links are another story.
        -- They are maintained by higher level tools and allow the link name
        -- to be different than the simple name of the designated object.

        -- ALL of these procedures are subject to a bad status resulting form
        -- the inability to acquire appropriate lock(s) or insufficient disk
        -- space on the unit.


        package Om renames Om_Definitions;

        subtype Context is Om.Naming_Context;


        procedure Create (In_World : Om.Version_Handle;
                          Link_Name : Om.Simple_Name_String;
                          Designated_Name : Om.Unique_Wildcard;
                          Id : Om.Action_Id;
                          Status : out Error.Condition;
                          In_Context : Context := Job.Default_Context;
                          Max_Wait : Duration := Job.Default_Wait);

        procedure Create (In_World : Om.Version_Handle;
                          Link_Name : Om.Simple_Name_String;
                          Designated_Object : Om.Object_Handle;
                          Id : Om.Action_Id;
                          Status : out Error.Condition;
                          Max_Wait : Duration := Job.Default_Wait);

        -- Usual rules for Naming.Resolve, and interpretation of the
        -- Version_Handle/Object_Handle, as specified by the Object package.

        -- Creates an external link.
```

```
        -- Acquires a Record_Write_Object lock on the link pack, and one or
        -- more Record_Write lock's within the link pack, with the given action
.

        -- The In_World parameter may refer to any object/version in the
        -- world of interest.

        -- Possible errors:
        --   Is_Link_Error (Status):
        --       Duplicate_Link_Name
        --           o The given Link_Name would duplicate that of an already
        --             existing internal or external link.
        --       External_Link_To_Local_Object
        --           o The Designated_Object is in the same world as the
        --             link pack.
        --       Cant_Create_External_Link
        --           o Insufficient internal data structures exist to create
        --             this link.
        --   Is_Bad_Version_Handle (Status)

      procedure Delete (In_World : Om.Version_Handle;
                        Link_Name : Om.Simple_Name_String;
                        Id : Om.Action_Id;
                        Status : out Error.Condition;
                        Max_Wait : Duration := Job.Default_Wait);

        -- Deletes an existing link.

        -- The In_World parameter may refer to any object/version in the
        -- world of interest.

        -- Acquire a Record_Write_Object lock on the link pack, and one or
        -- more Record_Write lock's within the link pack, with the given
        -- action.

        -- Possible errors:
        --   Is_Link_Error (Status):
        --       Link_Name_Does_Not_Exist
        --       Cant_Delete_Internal_Link
        --   Is_Bad_Version_Handle (Status)

      procedure Resolve (Link_Name : Om.Unique_Wildcard;
                         Designated_Object : out Om.Object_Id;
                         Status : out Error.Condition);

        -- The Link_Name is actually a Simple_Wildcard which must resolve to
        -- just a single link.  If Status is successful, then the given
        -- Link_Name maps to Designated_Object.

        -- Used by Naming.Resolve when it encounters a "*" character.

        -- Possible errors:
        --   Is_Bad_Pathname (Status)

      generic
          with procedure Visit (Result : Om.Object_Id);

      procedure Wild_Resolve (Link_Name : Om.Simple_Wildcard;
```

```
                                Status : out Error.Condition);

   -- Similar to the Resolve procedure. Is happy with 0 or more matches,
   -- calling Visit for each match. Stops searching after first error.

   -- Used by Naming.Wild_Resolve when it encounters a "*" character.

   procedure Get_Full_Pathname (Designated_Object : Om.Object_Id;
                                Full_Name : out Om.Pathname;
                                Status : out Error.Condition);

   -- Typically used when Resolve returns an Object_Id which turns out
   -- to be a dangling reference, and one wants to display a message
   -- indicating what we thought the link was supposed to reference.

   -- Possible errors:
   --   Is_Link_Error(Status):
   --        Cant_Identify_Link

   ------------------------------------------------------------------
   -- Use of the following operations is restricted to Object'body --
   ------------------------------------------------------------------

   procedure Differentiate (Link_Pack : Machine.Segment_Name;
                            Status : out Error.Condition);

   -- Called by Object.Open when undifferentiating a link pack.  The
   -- given segment is open with mode=Write_Object.  The operation
   -- copies the map which accelerates resolution of link names.

   procedure Create_Internal_Link
                  (Link_Pack : Om.Version_Handle;
                   Link_Name : Om.Simple_Name_String;
                   Designated_Object : Om.Object_Id;
                   Id : Om.Action_Id;
                   Status : out Error.Condition;
                   Max_wait : Duration := Job.Default_Wait);

   -- Creates an internal link.

   -- Acquires a Record_Write_Object lock on the link pack, with the
   -- given action (the same action as given to Object.Create).

   -- Link_Name must be the simple name of Designated_Object.

   -- Acquires the Version_Map Muxtex during the deletion of the link;
   -- No Log record is written; assumes Client will call
   -- Delete_Internal_Object if abandoning the action.

   -- Possible errors:
   --   Is_Link_Error(Status):
   --        Duplicate_Link_Name

   procedure Delete_Internal_Link
                  (Link_Pack : Om.Version_Handle;
                   Link_Name : Om.Simple_Name_String;
                   Id : Om.Action_Id;
                   Status : out Error.Condition;
```

```
                    Max_Wait : Duration := Job.Default_Wait);

      -- Deletes the internal link, with the given Link_Name.

      -- Acquires a Record_Write_Object lock on the link pack, with the
      -- given action (typically a private action of expunge).

      -- Acquires the Version_Map Muxtex during the addition of the link;
      -- No Log record is written; assumes Client will call
      -- Create_Internal_Object if abandoning the action.

      -- Possible errors:
      --   Is_Link_Error(Status):
      --         Link_Name_Does_Not_Exist

   end Link;

   ------------------------------------------------------------------
   ------------------------------------------------------------------

   package User is
      package Om renames Om_Definitions;

      -- Users are denoted by a directory (with subclass User_World) under
      -- "!Users". Various security related parameters are stored in
      -- "!Users._Profiles_". Passwords are stored in "!Users._Passwords_".
      -- Accounting information in stored in the "!Users.Accounting_Log".
      -- Sessions are simply directories with subclass Session, typically in
      -- the users home directory. Groups are denoted by a directory (with
      -- subclass Group) under "!Groups"; uses a link pac to construct the
      -- permanent representation of the set of users/groups in the group
      -- (denoted by the directory).


      procedure Create (User_Name : Om.Pathname_String;
                        Password : String;
                        Home_Directory : out Om.Object_Handle;
                        Users_Id : out Om.User_Id;
                        Status : out Error.Condition);

      -- Creates a new world for the user, returned in Home_Directory.
      -- Initially, User_Name must be a simple name. At some future point in
      -- time, might allow User_Name to also specify intervening directories
      -- between "!Users" and the new user's home directory.

      -- Possible errors:
      --   Is_Access_Control_Error (Status)

      procedure Get_Id (Name : Om.Unique_Wildcard;
                        Users_Object_Id : out Om.Object_Handle;
                        Users_Id : out Om.User_Id;
                        Status : out Error.Condition);

      procedure Get_Id (Name : Om.Object_Handle;
                        Users_Id : out Om.User_Id;
                        Status : out Error.Condition);

      -- With the first procedure, User_Name is resolved to an Object_Handle
```

```
      -- via Naming.Resolve. A resolve error implies the user does not
      -- exist. Then, with either procedure, computes the corresponding
      -- User_Id.

      -- Possible errors:
      --   Includes all those from Naming.Resolve
      --   Is_User_Error (Status)
      --       o Not_A_User_World
      --             o The Object_Handle does not identify a user world.

      function Simple_Name (User : Om.Object_Handle)
                  return Om.Simple_Name_String;

      -- Assuming User actually identifies an existing user, returns the
      -- user's simple name, as defined by Naming.Get_Simple_Name.

      function Home_Directory (User : Om.Object_Handle)
                  return Om.Pathname_String;

      -- Assuming User actually identifies an existing user, returns the
      -- full pathname of the user, as defined by Naming.Get_Full_Name.

      procedure Destroy (User : Om.Object_Handle;
                         Status : out Error.Condition;
                         Home_Directory_Must_Be_Empty : Boolean := False);

      -- The identified user will no longer be able to login. The user's
      -- home world is destroyed (in the sense defined by World.Destroy,
      -- supplying not Home_Directory_Must_Be_Empty for the
      -- Destroy_Contained_Objects and Destroy_Contained_Worlds parameters).

      -- Possible errors:
      --   All those returned by World.Destroy

      function Last_Login_Time (User : Om.Object_Handle) return Calendar.Time;
      function Last_Logout_Time
                  (User : Om.Object_Handle) return Calendar.Time;

      procedure Set_Password
                   (User : Om.User_Id;
                    Old_Password : String;
                    New_Password : String;
                    Status : out Error.Condition);

      -- Possible errors:
      --   Is_Access_Control_Error (Status)

      procedure Validate_Password
                   (User : Om.User_Id;
                    Password : String;
                    Status : out Error.Condition);

      -- Possible errors:
      --   Is_User_Error (Status)
      --       o Invalid_Password


      package Group is
```

```
        procedure Create (Name : Om.Pathname_String;
                          New_Id : Om.Group_Id;
                          Action : Om.Action_Id;
                          Status : out Error.Condition);
        procedure Get_Id (Name : Om.Unique_Wildcard;
                          Its_Id : out Om.Group_Id;
                          Status : out Error.Condition);
        procedure Delete (Group : Om.Group_Id;
                          Action : Om.Action_Id;
                          Status : out Error.Condition);

        package Member is
            procedure Add (Group : Om.Group_Id;
                           User : Om.User_Id;
                           Action : Om.Action_Id;
                           Status : out Error.Condition);
            procedure Remove (Group : Om.Group_Id;
                              User : Om.User_Id;
                              Action : Om.Action_Id;
                              Status : out Error.Condition);
        end Member;
    end Group;
end User;

-------------------------------------------------------------
-------------------------------------------------------------

package View is

    package Om renames Om_Definitions;

    -- We assume reader is already familiar with the basic meaning and
    -- function universe and world views.

    -- A universe view is thought of as an array of elements, indexed by
    -- world number. Each element is called a "slot".

    -- There are 3 values stored in a universe view slot:
    --      Bound_World_View      : Om.Version_Id;
    --      Bound_Wv_Is_Protected : Boolean;
    --      Indirect_World_View   : Om.Object_Id;
    -- and they have the following meaning:

    -- If Bound_World_View and Indirect_World_View are both Nil, then this
    -- slot is said to be Nil, and does not reference any world view.

    -- If Bound_World_View is not Nil, this slot is said to be "bound" and
    -- to reference the identified world view version. Note that the world
    -- number (in the version id) is always the same as the index (of the sl
ot).

    -- If Bound_World_View is Nil, this slot is said to be "indirect". The
    -- Indirect_World_View references the corresponding element in the
    -- identified universe view. In this case, one "resolves" an object id
    -- by following these indirect links until a bound slot is
    -- encountered. Cycles in the indirection are detected by having a
    -- maximum depth (of 31, say).
```

```
   -- If both Bound_World_View and Indirect_World_View are non Nil, the
   -- slot is said to be "refreshable", meaning that Bound_World_View can
   -- be updated to reference to whatever Indirect_World_View resolves.

   -- Bound_Wv_Is_Protected can only be true when Bound_World_View is not
   -- Nil. If Bound_Wv_Is_Protected is true, then the bound world view is
   -- protected from being expunged, as long as the universe view still
   -- exists.


   procedure Create (Target_Universe_View : Om.Unique_Wildcard;
                     Action : Om.Action_Id;
                     New_Universe_View : out Om.Version_Handle;
                     New_World_View : out Om.Version_Handle;
                     Status : out Error.Condition;
                     Crippled : Boolean := False;
                     Indirect : Om.Version_Handle := Om.Nil_Version_Handle;

                     Max_Ada_Objects : Integer := - 1;
                     Max_Other_Objects : Integer := - 1;
                     Version_Retention : Om.Version_Retention_Count := 1;
                     Max_Wait : Duration := Job.Default_Wait);

   -- This operation is typically used to initialize a new world, and
   -- when creating a new path from scratch.

   -- Max_Ada_Objects is interpreted only when the Indirect parameter is
   -- nil; similarly for Max_Other_Objects. In this case, if
   -- Max_Ada_Objects is negative, then it defaults to the value returned
   -- by World.Max_Ada_Objects (for the world in which the world view
   -- version is created). Similarly for Max_Other_Objects.

   -- Creates a new universe view, initialized as follows: it
   --    o  is unfrozen,
   --    o  all slots are nil, except the slot for the world in which the
   --       universe view is created (note that a crippled universe view
   --       has just this one slot).
   -- If the Indirect parameter is not nil, then the slot is set to be
   -- and indirect reference through the specified universe view. Note
   -- that no check is made to verify that the identified universe view
   -- actually exists. If the Indirect parameter is nil, then this slot
   -- references a new version of the world view, which is initialized as
   -- follows: it
   --    o  is unfrozen
   --    o  is not protected
   --    o  can support the specified number of Ada and other objects
   --    o  each slot can reference the specified number of versions
   --    o  all slots are nil except the slot for the link pack, which
   --       references a new version of the link pack, which is empty.
   --    o  all slots have the Reserved bit set
   --    o  all unit states are nil
   --    o  the dependency matrix is clear.

   -- Note that the new universe view is NOT entered into any world
   -- views.

   -- If Target_Universe_View is "", the simple name of the new universe
   -- view is computed using the same rules as for creating temporary
```

```
        -- object names in Object.Create (with ""); the universe view will be
        -- a child of the world in which it is created.

        -- The new universe and world view are both held with Write_Object
        -- locks. But note that the Object operations and the universe view
        -- slot operations (below) still work on the view (even though upgrade
        -- from WO to RWO is normally not supported).

        -- Destroy of a Universe_View/World_View can be acheived via
        -- Object.Expunge.

        -- Possible errors:
        --   Is_Bad_Pathname (Status)
        --   Is_View_Error (Status)
        --       o Cant_Recreate_View
        --             The object named by Target_Universe_View already exists.
        --       o Bad_Object_Index_Constraint
        --             The value of Max_Ada_Objects or Max_Other_Objects is not
        --             within the bounds prescribed by the parameters of the
        --             containing world.
        --   Is_Bad_Action (Status)
        --   Is_Lock_Error (Status)
        --   Is_Resource_Limit_Error (Status)


        type Freeze_Option is new Long_Integer range 0..7;

        Same_Frozenness_As_Source : constant Freeze_Option := 1;
        Freeze_Target             : constant Freeze_Option := 2;
        Unfreeze_Target           : constant Freeze_Option := 3;

        Default_Res_Bits          : Om.Reservation_Bits (1..0);

        procedure Copy (Source_Universe_View : Om.Version_Handle;
                        Target_Universe_View : Om.Unique_Wildcard;
                        Action : Om.Action_Id;
                        Status : out Error.Condition;
                        Version_Prune_Target : Boolean := False;
                        This_World : Freeze_Option := Same_Frozenness_As_Source;

                        Other_Worlds : Freeze_Option := Same_Frozenness_As_Sourc
e;

                        Ada_Res_Bits : Om.Reservation_Bits := Default_Res_Bits;
                        Other_Res_Bits : Om.Reservation_Bits := Default_Res_Bits
;

                        Max_Ada_Objects : Integer := - 1;
                        Max_Other_Objects : Integer := - 1;
                        Max_Wait : Duration := Job.Default_Wait);

        -- This operation can be used to implement higher level operations
        -- such as fork a path, spawn a subpath, make a release, vanilla view
        -- copy, etc.

        -- The target universe view is created, and initialized as follows:
        -- its frozen'ness is as specified by This_World Freeze_Option.
        -- parameter. It has the same Crippled'ness as the source. For empty
        -- slots in the source universe view, the target universe view has
        -- empty slots. For non-empty slots in the source, the corresponding
```

```
    -- target slot is initialized as follows:

    -- If the source slot is indirect, the target slot becomes identical
    -- to the source slot.  But note that this is no guarantee that the
    -- indirection actually leads to someplace interesting!

    -- If the source slot is bound (the only remaining case), the
    -- following rules apply:

    -- The target slot will reference the same world view (as the source)
    -- only if the source world view is frozen and not protected by the
    -- source universe view.

    -- In all other cases, a new world view version is created, which is
    -- copy of the source world view, except as indicated by the following:

    -- If the target world view is in the same world as the target
    -- universe view, then the This_World parameter applies, else the
    -- Other_Worlds parameter applies.  The target world view will be
    -- frozen iff
    --    (a) The Freeze_Option is Same_Frozenness_As_Source, and the
    --        source world is frozen, or
    --    (2) The Freeze_Option is Freeze_Target.

    -- The target world view will be version pruned if the target world
    -- view is to be frozen, or the Version_Prune_Target parameter is
    -- true. By "version pruned" we mean that the retention lists in the
    -- new world view will be of length 1 (may expunge deleted versions).

    -- The target world view will be object pruned if the target world
    -- view is to be frozen. By "object pruned" we mean that the maximum
    -- number of Ada units and other objects will be reduced to cover just
    -- the objects that are actually selected by the world view (makes the
    -- view smaller).

    -- The target world view will be protected by the target universe view
    -- if the source world view is protected by the source universe view.

    -- If Max_Ada_Objects is negative, then it defaults to the value
    -- returned by World.Max_Ada_Objects (for the world in which the
    -- target world view version is created). Similarly for
    -- Max_Other_Objects.

    -- If the target world view is unfrozen, then the following apply: (1)
    -- The target view can support the number of Ada and other objects
    -- specified by the Max_Ada_Objects and Max_Other_Objects parameters.
    -- (2) If both Ada_Res_Bits and Other_Res_Bits are null arrays, then
    -- the reserved bits in the target world view are identical to those
    -- of the source world view. Otherwise, the reserved bits (in the
    -- target world view) are set as specified by the value of the
    -- Ada_Res_Bits and Other_Res_Bits parameters; objects not specified
    -- have their reserved bits True; objects which appear in both the
    -- Ada_Res_Bits and Other_Res_Bits get an unspecified value for their
    -- reserved bit; the operation ignores slices of Ada_Res_Bits or
    -- Other_Res_Bits which are out of the range of Object_Indexes defined
    -- for the view.

    -- The new universe and world views are held with write_Object locks.
```

```
   -- But note that the Object operations and the universe view slot
   -- operations (below) still work on the views (even though upgrade
   -- from WO to RWO is normally not supported).

   -- Note that link packs are differentiated on demand, just like
   -- versions of other objects; therefore no special rules are required.

   -- Possible errors:
   --   Is_Bad_Version_Handle (Status)
   --       o The Source_Universe_View does not exist
   --   Is_Bad_Pathname (Status)
   --   Is_View_Error (Status)
   --       o Cant_Recreate_View
   --             The object named by Target_Universe_View already exists.
   --       o Bad_Object_Index_Constraint
   --             The value of Max_Ada_Objects or Max_Other_Objects is not
   --             within the bounds prescribed by the parameters of the
   --             containing world.
   --   Is_Bad_Action (Status)
   --   Is_Lock_Error (Status)
   --   Is_Resource_Limit_Error (Status)


   procedure Get_Universe_View_Slot
                  (Universe_View : Om.Version_Handle;
                   Action : Om.Action_Id;
                   Bound_World_View : out Om.Version_Handle;
                   Bound_wv_Is_Protected : out Boolean;
                   Indirect_World_View : out Om.Version_Handle;
                   Status : out Error.Condition;
                   Max_Wait : Duration := Job.Default_Wait);

   procedure Set_Universe_View_Slot
                  (Universe_View : Om.Version_Handle;
                   For_World : Om.Object_Handle;
                   Action : Om.Action_Id;
                   Bound_World_View : Om.Version_Handle;
                   Bound_wv_Is_Protected : Boolean;
                   Indirect_World_View : Om.Version_Handle;
                   Status : out Error.Condition;
                   Max_Wait : Duration := Job.Default_Wait);

   -- These operations are typically used to change "imports".

   -- The get operation reads all 3 values from the identified slot. The
   -- meaning of these values is defined at the beginning of this
   -- package. The set operation writes all 3 values into the identified
   -- slot.  Client can combine the 2 operations to create operations
   -- which change just a single value.

   -- For_World identifies some object in the world whose corresponding
   -- slot (in Universe_View) is to be read/written.

   -- The get operation acquires a Record_Read lock on the identified
   -- slot. The set operation acquires a Record_Write lock on the slot
   -- (or a Write_Object lock if the action already holds a read/write
   -- object lock).
```

```
        -- Possible errors:
        --    Is_Bad_Version_Handle (Status)
        --    Is_View_Error (Status)
        --        o World_View_Not_Bound
        --              Attempting to set Bound_Wv_Is_Protected, but
        --              Bound_World_View is Nil.
        --    Is_Bad_Action (Status)
        --    Is_Lock_Error (Status)
        --    Is_Resource_Limit_Error (Status)

        procedure Expunge (Universe_View : Om.Version_Handle;
                           Number_Expunged : out Natural;
                           Disk_Blocks_Freed : out Natural;
                           Status : out Error.Condition);

        -- Calls Object.Expunge on every object selected by the world view
        -- (selected by Universe_View) in the same world as Universe_View.

        -- On return, Number_Expunged and Disk_Blocks_Freed are returned equal
        -- to the sum of the values returned by the calls to Object.Expunge.

        -- Also, cleans up some internal OM garbage. Specifically: (1) Due to
        -- outstanding locks at expunge time, it is possible for versions to
        -- exist and not be selected by any view; these versions are expunged.
        -- (2) Also due to outstanding locks at expunge time, it is possible
        -- for expungeable objects with no versions to exist; these objects
        -- are expunged. (3) If an object goes from having 0 versions to 1 or
        -- more versions, an empty Nil version is left around (invisible to
        -- users); these versions are expunged. In future implementations,
        -- these sources of garbage may be eliminated, and this paragraph
        -- stricken.

        -- Possible errors:
        --    Is_Bad_Version_Handle (Status)
        --    plus all those returned by Object.Expunge.

        procedure Destroy (View : Om.Version_Handle;
                           Number_Expunged : out Natural;
                           Disk_Blocks_Freed : out Natural;
                           Status : out Error.Condition);

        -- Class_World_View: For a world view which is no longer referenced,
        -- this operation will destroy the identified version of the "_View_"
        -- object and expunge any version (and possibly object) which is
        -- no longer referenced as a result.
        -- (Except that the "_View_" and "_Links_"
        -- objects are never expunged).

        -- Class_Universe_View: This operation will destroy the identified
        -- universe view and destroy any world view versions (as defined
        -- above) which are no longer referenced as a result.

        -- Possible errors:
        --    Version_Is_Still_Referenced
        --        o Attempt to destroy a world view version which is still
        --          referenced by some universe view.

    end View;
```

```
    ------------------------------------------------------------
    ------------------------------------------------------------

    package World is
        -- pragma Subsystem (Object_Management)

        package Om renames Om_Definitions;

        subtype Version_Retention_Count is Om.Version_Retention_Count;

        subtype Context is Om.Naming_Context;

        ----------------------
        -- Basic operations --
        ----------------------

        function Default_Unit return Disk.Unit_Number;

        -- Returns a legal Unit_Number according to a policy that is currently
        -- unspecified. Probably a policy similar to "best/usable volume" in
        -- Gamma.

        function Default_Cluster_Size return Disk.Block_Count;
        -- Currently 1.

        function Default_Warning_Threshold return Disk.Region_Count;
        -- Currently infinite.

        function Default_Allocation_Increment return Disk.Block_Count;
        -- Currently 1 track.


        procedure Create (Name : Om.Unique_Wildcard;
                          Max_Ada_Objects : Positive;
                          Max_Other_Objects : Positive;
                          New_World : out Om.Object_Handle;
                          Status : out Error.Condition;
                          In_Context : Context := Job.Default_Context;
                          Unit : Disk.Unit_Number := Default_Unit;
                          Cluster_Size : Disk.Block_Count := Default_Cluster_Siz
e;
                          Warning_Threshold : Disk.Block_Count := Default_Warnin
g_Threshold;
                          Allocation_Increment : Disk.block_Count := Default_All
ocation_Increment);

        -- Name and In_Context are used to specify the name of the new world,
        -- in a fashion similar to Object.Create.

        -- The following objects are created (as children of the world
        -- object): The object with simple name "_Views_" has
        -- Class_World_View; versions of this object are used to construct
        -- various "views" of the world.  The object with simple name
        -- "_Links_" has Class_Link_Pack; versions of this object are selected
        -- by versions of the world view in order to represent the Ada library
        -- unit name space.
```

```
        -- The values of Max_Ada_Objects and Max_Other_Objects control the
        -- size of a table (of object information) which is stored in the
        -- world object. If one specifies maximal values you chew up something
        -- like a 1/3 of a Mbyte in the world object. These values constrain
        -- the corresponding values when creating new views (via the
        -- View.Create and View.Copy).

        -- Some additional objects are created (as children of the world
        -- object): The object with simple name "_Initial_" has
        -- Class_Universe_View and is created by calling View.Create with the
        -- appropriate string name and default parameters. The object with
        -- simple name "_Default_" has Class_Universe_View and is created by
        -- calling View.Create with the appropriate string name, Indirect to
        -- the "_Initial_" view, Crippled, and remaining parameters taking
        -- defaults.

        -- The creation takes permanent effect immediately - there is no
        -- action involved.

        -- Note that the created world version is not put in any world view,
        -- and there is no way to put it into one.

        -- The System_View is updated to contain an indirect reference to the
        -- "_Default_" view created for this world.

        -- Possible errors:
        --   Is_Bad_Pathname (Status):
        --        o Might be because the last segment of the name contains
        --          wildcards.
        --   Illegal_Class
        --        o The "'C" attribute specified something other than a director
y.
        --   Is_Resource_Limit_Error (Status)
        --        o Like ran out of world numbers.

        procedure Destroy (Name : Om.Unique_Wildcard;
                           Status : out Error.Condition;
                           In_Context : Context := Job.Default_Context;
                           Destroy_Contained_Objects : Boolean := False;
                           Destroy_Contained_Worlds : Boolean := False);

        procedure Destroy (The_world : Om.Object_Handle;
                           Status : out Error.Condition;
                           Destroy_Contained_Objects : Boolean := False;
                           Destroy_Contained_Worlds : Boolean := False);

        -- Name, and In_Context are used according to the rules of
        -- Naming.Resolve to compute the Object_Id of the world to be destroyed.

        -- All of the regions allocated to this world are made available to
        -- the pool of free regions.  This operation takes permanent effect
        -- immediately. This operation DOES NOT follow action semantics. So,
        -- once the world is destroyed there is no way to get the objects back.

        -- WARNING: Supplying True to either of the destroy options will cause
        -- the objects which live in the world to be destroyed.

        -- This operation attempts to produce its errors prior to actually
```

```
        -- destroying anything.  However, the destroy operation is not atomic
        -- with respect to concurrent operations (like Object.Create);
        -- consequently, it is possible to get errors after contained objects
        -- have been destroyed.

        -- The entry (for this world) in the System_View is removed.

        -- Possible errors:
        --    Is_Bad_Object_Id (Status)
        --    Is_Bad_Object_Handle (Status)
        --    Is_Bad_Pathname (Status)
        --    Is_Bad_Action (Status)
        --    Is_Lock_Error (Status)
        --    Version_Is_Still_Referenced
        --        o Destroy_Contained_Objects is true, but the (sub) world
        --          contains at least one version which cannot be expunged.
        --    World_Still_Contains_Objects
        --        o Destroy_Contained_Objects is false, but the world contains
        --          objects (with unexpunged versions) other than the link pac
        --          and world view.
        --    World_Still_Contains_Subworlds
        --        o Destroy_Contained_Objects is false, but the world contains
        --          objects which are themselves worlds.

        function System_View return Om.Version_Handle;

        -- Returns the id of the "system view".

        procedure Expunge (World : Om.Unique_Wildcard;
                           Number_Expunged : out Natural;
                           Disk_Blocks_Freed : out Natural;
                           Status : out Error.Condition);

        procedure Expunge (World : Om.Object_Handle;
                           Number_Expunged : out Natural;
                           Disk_Blocks_Freed : out Natural;
                           Status : out Error.Condition);

        -- Calls View.Expunge (not Object.Expunge!) on every universe view in
        -- the identified world.  Number_Expunged and Disk_Blocks_Freed are
        -- returned equal to the sum of the values returned by the calls to View
.Expunge.

        -- Possible errors:
        --    Illegal_Class
        --        o Can only be applied to an object with Class_Directory and
        --          subclass world.
        --    All those returned by View.Expunge.


        procedure Move (Name : Om.Unique_Wildcard;
                        To_Unit : Disk.Unit_Number;
                        Status : out Error.Condition;
                        In_Context : Context := Job.Default_Context);

        procedure Move (World : Om.Object_Handle;
                        To_Unit : Disk.Unit_Number;
```

```
                    Status : out Error.Condition;
                    In_Context : Context := Job.Default_Context);

   -- Name and In_Context are used according to the rules of
   -- Naming.Resolve to compute the Object_Id of the world to be moved.
   -- Moves the specified world to the specified disk. Requires that
   -- there be no locks of any kind on any object in the world. Ok if the
   -- world is already on that disk. Atomic with respect to concurrent
   -- activity. Atomic with respect to crashes. Probably not available in
   -- the 1st qtr implementation.

   -- Possible errors:
   --    Illegal_Class
   --        o Can only be applied to an object with Class_Directory and
   --          subclass world.
   --    Is_Bad_Object_Id (Status)
   --    Is_Bad_Pathname (Status)
   --    Is_Bad_Action (Status)
   --    Is_Lock_Error (Status)
   --    Is_Resource_Limit_Error (Status)

   ----------------------
   -- world parameters --
   ----------------------

   -- These functions return garbage (typically 0) when given garbage
   -- input parameters.  Procedures which do not return status are noops
   -- when the input parameters are garbage. The set operations take
   -- permanent effect immediately. These operations take a World
   -- parameter, which can identify any object in the world of interest.

   function Cluster_Size
                  (World : Om.Object_Handle) return Disk.Block_Count;

   -- Returns the cluster size for this world.  Currently, we only
   -- support a cluster size of 1.  It is not possible to change the
   -- cluster size once a world has been created.

   function Warning_Threshold (World : Om.Object_Handle)
                  return Disk.Block_Count;
   procedure Set_Warning_Threshold (World : Om.Object_Handle;
                                    Threshold : Disk.Region_Count);

   -- Offending sessions get a warning message when the number of
   -- allocated regions (in this world) goes above this value. Default:
   -- Region_Count'Last.

   function Max_Header_Allocation_Increment return Disk.Block_Count;

   function Object_Header_Allocation_Increment
                  (World : Om.Object_Handle) return Disk.Block_Count;
   procedure Set_Object_Header_Allocation_Increment
                  (World : Om.Object_Handle;
                   Increment : Disk.Block_Count);

   -- To minimize crash recovery time, it is advantageous for object
   -- headers to be allocated contiguously on the disk. When a new object
   -- header must be allocated, and the world does not contain any free
```

```
        -- object headers, the system allocates N blocks for use as object
        -- headers, where N is the "allocation increment" parameter controlled
        -- by these 2 operations. Larger values of the allocation incrment
        -- tend to increase contiguity, but also increases fragmentation
        -- (since free object header blocks cannot be used for anything else).

        -- The value must not exceed Max_Header_Allocation_Increment.

        function Max_Ada_Objects (World : Om.Object_Handle) return Positive;
        procedure Set_Max_Ada_Objects (World : Om.Object_Handle;
                                       Max_Ada_Objects : Positive;
                                       Status : out Error.Condition);

        function Max_Other_Objects (World : Om.Object_Handle) return Positive;
        procedure Set_Max_Other_Objects (World : Om.Object_Handle;
                                         Max_Other_Objects : Positive;
                                         Status : out Error.Condition);

        -- As indicated under the Create operation, the value of these
        -- parameters constrains the corresponding values in View.Create and
        -- View.Copy operations.

        -- Possible errors:
        --     Is_View_Error (Status)
        --         o Bad_Object_Index_Constraint
        --               Can be returned for any of the following: The given
        --               value of Max_Ada_Objects is smaller than the correspond
ing
                                                                 ,
        --               value for some version of the world view for this world
.
        --               Similar situation for Max_Other_Objects. The sum of
        --               Max_Ada_Objects and Max_Other_Objects exceeds the maxim
um
        --               number of object indices.

        -----------------------
        -- World information --
        -----------------------

        -- These operations return garbage when the supplied World_Number
        -- refers to a world which does not exist.  These operations take a
        -- World parameter, which can identify any object in the world of
        -- interest.

        function Unit (World : Om.Object_Handle) return Disk.Unit_Number;
        function Is_A_Job_World (World : Om.Object_Handle) return Boolean;

        function Consumed_Regions (World : Om.Object_Handle)
                     return Disk.Region_Count;

        -- Returns the number of regions which are currently allocated to
        -- world.

        subtype Block_Count is Disk.Block_Count;

        function Block_Capacity (World : Om.Object_Handle) return Block_Count;
        function Header_Capacity (World : Om.Object_Handle) return Block_Count;
        function Data_Block_Capacity
```

```
                    (World : Om.Object_Handle) return Block_Count;
    function Consumed_Headers (World : Om.Object_Handle) return Block_Count;
    function Consumed_Data_Blocks
                    (World : Om.Object_Handle) return Block_Count;
    function Structural_Map_Size
                    (World : Om.Object_Handle) return Block_Count;
    function Version_Map_Size (World : Om.Object_Handle) return Block_Count;

    -- Block_Capacity is simply Consumed_Regions times the number of
    -- blocks in a region.  Header_Capacity is the number of blocks
    -- currently reserved for use as headers, and Data_Block_Capacity is
    -- the number of blocks remaining for other uses.  Block_Capacity less
    -- Header_Capacity and Data_Block_Capacity identifies the built in
    -- overhead of a world. Consumed_Headers is the number of header
    -- blocks currently in use. Similarly, Consumed_Data_Blocks is the
    -- number of blocks currently in use for other things.  Each of the
    -- following expressions represent different kinds of internal
    -- fragmentation:
    --    (1) Header_Capacity - Consumed_Headers
    --    (2) Data_Block_Capacity - Consumed_Data_Blocks
    --    (3) (Header_Capacity + Data_Block_Capacity) -
    --           (Consumed_Headers + Consumed_Data_Blocks)
    -- Structural_Map_Size and Version_Map_Size represent the only other
    -- forms of overhead in the world which is not accounted for by
    -- looking at the amount of space consumed by individual versions of
    -- objects.


    ----------------------------------------
    -- Window of Vulnerability Operations --
    ----------------------------------------

    -- These operations are noops when given invalid World_Number's.

    function Is_Vulnerable (World : Machine.World_Number) return Boolean;
    procedure Make_Vulnerable (World : Machine.World_Number);
    procedure Make_Invulnerable (World : Machine.World_Number);

    -- A world enters its window of vulnerability when its contents are
    -- modified. Vulnerable worlds have to be examined, object by object,
    -- by crash recovery. Thus, causing a world to leave its window of
    -- vulnerability may reduce the cost of crash recovery. There is a
    -- daemon which attempts to use idle cycles to remove worlds from
    -- their window of vulnerability.

end World;

------------------------------------------------------------------
------------------------------------------------------------------

package Access_Control is

    -- pragma Subsystem (Object_Management);

    -- The following is a summary of proposed access control rules.

    ------------------
    -- General rules --
```

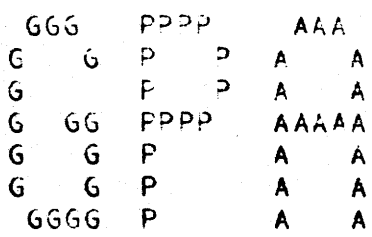
```
------------------

-- Conventional definition for user. Max 1023 users. User "George"
-- exists iff there is an object with name "!Users...George", where
-- "!Users" is a predefined world, "George" is a direct child world of
-- "!Users", and "..." can be an intervening chain of 0 or more
-- non-world directories. By default, only the Privileged group has
-- the ability to create/delete users. OM will not allow one to create
-- any other objects in the "!Users" world.

-- Conventional definition for group. Max 4095 groups. Group
-- "Software" exists iff there is an object with name
-- "!Groups.Software", where "!Groups" is a predefined world, and
-- "Software" is a directory with Group_Subclass. George is a member
-- of Software iff there is a directory with the name
-- "!Groups.Software.George", with Member_Subclass. By default, only
-- the Privileged group has the ability to create/delete/modify
-- groups. OM will not allow one to create any other objects in the
-- "!Groups" world.

-- Every user is implicitly a group. Implicitly defined group Public
-- includes all users on the machine. Implicitly defined group
-- Network_Public includes all users with account on remote machines.

-- Predefined users Operator and Rational.

-- Predefined group Privileged includes users Operator and Rational.
-- Can use basic mechanisms to include arbitrary users in the
-- Privileged group. The Privileged group is special in the sense that
-- the privileges conferred by being a member of this group have to be
-- explicitly enabled. When Privileged capabilities are enabled, there
-- are no access control restrictions in force.

-- As indicated by the Job package, each session/job has an associated
-- "owner" user id. In addition, there is an amplification list of up
-- to 3 user ids. The rights of the session/job are the union of the
-- rights of the owner and the users named in the amplification list.
-- Recall that jobs are initialized to have this state equal to that
-- of the creating session.

-- ACLs are defined in OM_Definitions, and should be fairly obvious.
-- ACLs are applied to individual versions. There is no object-level
-- ACL.

-- The following privileges are defined:
--   Create
--   Delete
--   Read
--   Write
--   Execute
--   Owner

-- The specific meaning of the privileges is defined below:
--
--   Create       Applies only to worlds and controls the ability to
--                create views and direct world descendents.
--
--   Delete       Applies only to worlds and views. For worlds,
```

```
--                  controls the ability to destroy this world.  For
--                  views, controls the ability to destroy the view
--                  itself.
--
--    Read          Applicable to all objects except directories (and
--                  therefore worlds), and link packs. For world views,
--                  controls the ability to open objects, via the view.
--                  For other objects, controls the ability to open the
--                  specific version for reading.
--
--    Write         Applicable to all objects except directories (and
--                  therefore worlds).  For world views, controls the
--                  ability to modify the view; this includes
--                  creation/deletion of objects/versions via the view.
--                  For other objects, controls the ability to open the
--                  specific version for modification.
--
--    Execute       Applicable only to load images and Ada units/images.
--                  Further discussion below.
--
--    Owner         Applicable only to worlds. Controls the ability to
--                  change ACLs on versions stored in the world, as well
--                  as other miscellaneous things.

-- Note that worlds have an associated "owner" which is used for
-- accounting. The world's owner need not have Owner access granted in
-- the ACL of the world object (although this would certainly be true
-- by default).

-- World views contain a default ACL for new objects/versions created
-- in this view. Recall that objects/versions (except for views and
-- worlds) cannot be created without doing it in a view. The world
-- object contains defaults for the ACL of views as well as for the
-- default ACL contained in the view.

-- Initial ACLs:
-- The ACL of a new version of an object which is currently "deleted"
-- from the perspective of the view is set from the view's default
-- ACL; this also covers the case where the object is being created.
-- The ACL of a new version (of an object which already "exists" from
-- the perspective of the view) is identical to that of the
-- predecessor version.

----------------------
-- World Operations --
----------------------

-- Create:
--       Requires create access to the parent world.

-- Destroy:
--       Requires destroy world access to the world being destroyed.

-- Move: Requires owner access to the world being moved.

-- Read world attributes:
--       Does not require any access.
```

```
        -- Write world attributes:
        --          Requires owner access to the world.

        -- Change ACL:
        --          Requires owner access to the world.

        -----------------------
        -- Naming Operations --
        -----------------------

        -- No access control is applied.

        -----------------------
        -- Object Operations --
        -----------------------

        -- Create:
        --      (1) To create a new object requires write access to the view.
        --          The new version will be writeable by the creator only if the
        --          default ACL in the view so specifies.
        --      (2) To create a new version of an object requires write access to
        --          the view. The new version will be writeable by the creator
        --          only if the ACL of the predecessor version so specified.

        -- Open:    Opening an object for read via a view requires read
        --          access to the view. Opening an object for update via a view
        --          requires write access to the view, regardless of whether or
        --          not differentiation is actually required. Note that when
        --          differentiation is performed, the new version will be
        --          writeable by the creator only if the ACL of the predecessor
        --          version so specified. Recall that one can open a version
        --          without going through a view by simply specifying the
        --          correct version number.  Also recall that many interesting
        --          classes of objects (such as Ada units) have a version policy
        --          which prohibits opening the version for update without going
        --          through a view. The ACL of the version itself must grant the
        --          desired access (read/write).

        -- Delete:
        --          Requires write access to the view.

        -- Undelete:
        --          Requires write access to the view.

        -- Expunge:
        --          Requires write access to the view.

        -- Copy:    Requires write access to the target view. If the version is
        --          being copied by reference, no other acess is required.
        --          Otherwise, read access to the source view and/or version is
        --          also required.

        -- Read version attributes:
        --          Requires read access to the version.

        -- Write version attributes:
        --          Requires write access to the version. This rule does not
        --          apply to change of ACL.
```

```
-- Change ACL:
--         Requires owner privilege to the enclosing world. Does not
--         require write access to the version itself.

---------------------
-- View Operations --
---------------------

-- Create:
--         Requires create access to the containing world.

-- Copy:  Requires read access to the source views, and create access
--         to the containing world.

-- Read/Write Univ view:
--         Requires read/write access to the universe view object.

-- Expunge:
--         Requires write access to the view.

-----------
-- Links --
-----------

-- The ACL for a link pack has no effect when creating/deleting
-- internal links or resolving link names. The ACL for a link
-- pack only has effect when creating/deleting external links.
-- Note that ACLs for link packs are inherited according to the
-- same rules as for other objects.

----------
-- Jobs --
----------

-- Various operations, such as setting state and job.kill require
-- requestor to have a group map which includes the owner of the
-- job/session in question.

---------------
-- Execution --
---------------

-- Elaborate program/subsystem:
--     (1) Importing code segments named by the load image requires
--         the executing job to have Execute access to the load image.
--     (2) Importing module names from already elaborated subsystem
--         requires (a) load image for imported subsystem enables
--         module access check, and (b) the executing job has Execute
--         access to the load image.

-------------------
-- Amplification --
-------------------

--     (1) Job/session can amplify its rights by supplying the password
--         of the added user.
--     (2) The Privileged group does not normally give its members any
```

```
        --          additional capabilities. A Job/Session can enable privileges
        --          if user is a member of the Privileged group.
        --      (3) Amplify_rights bit in primary load image will add rights of
        --          user, identified by the load image, to the elaborating job;
        --          rights go away when elaboration finished; prevents more than
        --          1 main program or subsystem from being elaborated by the
        --          job; setting the amplification rights requires the executing
        --          job to have the identified rights.
        --      (4) Can acheive amplification by elaborating subsystem in its
        --          own job, and running called procedure's on tasks of the
        --          elaborated subsystem.

        ---------------
        -- Scenarios --
        ---------------

        -- Strict partition:
        --          No problem other than one can "see" (in the Naming sense) the
        --          existence of controlled objects.

        -- Give read-only access to subsystem importers:
        --          No problem.

        -- Give Fred edit access to only a set of Ada units in a world:
        --          We assume that the Create/Destroy/Owner privileges (for the
        --          world) are not already given to Fred. Spawn a new view. Give
        --          Fred Read/Write access to the view. Set the view's default
        --          ACL to give Fred Read/Write access to units he creates. For
        --          all existing versions (in Fred's view) set the Read/Write
        --          privileges in the version's ACL) appropriately. Only modify
        --          versions whose ACLs actually need changing. Versions which
        --          get their ACLs changed are automatically differentiated. Fred
        --          can now edit selected units in the world. Fred might be able
        --          to install/code (some issues with ACLs on central databases
        --          kept by tools).

        -- Give Fred write access to a file in an unmanaged world, but prevent
        --          Fred from deleting the file. Can do it provided Fred is not
        --          allowed to create new versions.

        -------------------------
        -- Miscellaneous Notes --
        -------------------------

        -- Note that read/write privileges in the universe view are NOT
        -- checked during Object operations, such as Create and Open. This is
        -- true even when there is indirection in the views.

        -- Note that link packs have no ACL.

    end Access_Control;

end Om_Services_1;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.KK.OM_DEFINITIONS
Version: V(117)
Request: 1276

Date: April 24, 1986
Queued: 11:27:10 AM
Printed: 11:43:09 AM

```
with Sys;
with Filler;
with Bounded;
with Machine;

package Om_Definitions is
    -- pragma Subsystem (Kernel);

    -- General guidelines for use of OM specs:

    -- (1) Unless explictly stated otherwise, no operation propagates
    -- exceptions.


    package System renames Sys;

    Nil : constant := 0;

    -- All visibly scalar types have 0 as their Nil value.

    -----------------
    -- Machine ids --
    -----------------

    type Machine_Id is new Long_Integer range 0..2 ** 24 - 1;

    -- Guaranteed unique for all Rational machines.  Stored in the NOVRAM.

    --------------------
    -- Unique numbers --
    --------------------

    type Boot_Number is new Long_Integer range 0..2 ** 16 - 1;

    -- The system boot number is incremented each time the virtual memory
    -- system tries to boot itself.  This value is stored in the label of the
    -- root disk unit.  It is initialized when that disk unit is
    -- formatted/replaced.  The value is allowed to "wrap". Under normal use,
    -- this means that the system boot number is unique and monotonically
    -- increasing.

    type Unique_Sequence_Number is new Long_Integer range 0..2 ** 32 - 1;

    -- Starts at 1 each time the system boots.  Incremented each time the
    -- function Get is invoked.

    type Unique_Id is new Long_Integer range 0..2 ** 48 - 1;
    -- record
    --     Boot     : Boot_Number;
    --     Sequence : Unique_Sequence_Number;
    -- end record;

    function Boot return Boot_Number;
    function To_Unique_Id
                  (Boot : Boot_Number;
                   Sequence : Unique_Sequence_Number) return Unique_Id;
    function Boot (Id : Unique_Id) return Boot_Number;
    function Sequence (Id : Unique_Id) return Unique_Sequence_Number;
```

```
   function Get return Unique_Id;

   -- Within this run of the system, guarantees that the returned value
   -- will be ">" than all those returned by previous calls (even in the
   -- face of concurrent calls to Get).  With high probability,
   -- this property is also true in the case of crashes and recovery
   -- (from backup).  Costs roughly 3-4 usec to call this function.
   -- Concurrent calls are ok.  Calls by abortable threads are ok.

   -- At a consumption rate of 1 id every msec, will run out of Id's in
   -- 49 days, at which point the system crashes.

   type Net_Unique_Id is private;
   -- record
   --    Machine  : Machine_Id;
   --    Boot     : Boot_Number;
   --    Sequence : Unique_Sequence_Number;
   -- end record;

   Nil_Net_Unique_Id : constant Net_Unique_Id;

   function To_Unique_Id (Machine : Machine_Id;
                          Boot : Boot_Number;
                          Sequence : Unique_Sequence_Number)
               return Net_Unique_Id;
   function Extract_Machine (Id : Net_Unique_Id) return Machine_Id;
   function Boot (Id : Net_Unique_Id) return Boot_Number;
   function Sequence (Id : Net_Unique_Id) return Unique_Sequence_Number;

   function Get return Net_Unique_Id;

   -- Get the same properties as the previous Get function.  Plus the
   -- additional property that with high probability, the returned value is
   -- unique among the values generated by all Rational machines.

   ---------------------------------------------
   -- Object_Id, Version_Id and Object_Handle --
   ---------------------------------------------

   type Class_Number is new Long_Integer range 0..31;

   -- Uniquely identifies the "type" of data contained in the object.  For
   -- example, an object containing a Diana tree probably has a different
   -- type than an object containing a text file.  All versions of an object
   -- have the same class.

   Class_Directory          : constant Class_Number := 1;
   Class_World_View         : constant Class_Number := 2;
   Class_Universe_View      : constant Class_Number := 3;
   Class_File               : constant Class_Number := 4;
   Class_Pipe               : constant Class_Number := 5;
   Class_Link_Pack          : constant Class_Number := 6;
   Class_Ada                : constant Class_Number := 7;
   Class_Ada_Attributes     : constant Class_Number := 8;
   Class_Code               : constant Class_Number := 9;
   Class_Load_Image_Object  : constant Class_Number := 10;
   --                       : constant Class_Number := 11;
```

```
    --                          : constant Class_Number := 12;
    --                          : constant Class_Number := 13;
    Class_Tape                  : constant Class_Number := 14;
    Class_Terminal              : constant Class_Number := 15;

    -- The parent-child relationship makes most sense when the parent is
    -- a directory or Ada unit.  However, any object, regardless of class,
    -- may take the role of parent in the directory system.

    -- All classes of object may have multiple versions, except:
    --    World, Directory, and Device_Classes.

    type Subclass_Number is new Long_Integer range 0..2 ** 16 - 1;

    -- Rational reserves the first 1024 and provides a service for assigning
    -- subclasses to independent software vendors.  The Nil subclass indicates
    -- that there is no special interpretation for the object.  Examples:

    -- Subclasses for Class_Directory: World, Path ...
    -- Subclasses for Class_File: Text, Switches, Editor_Macros, ...
    -- Subclasses for Ada: Generic_Procedure_Spec, Package_Body, ...

    type Object_Index is new Long_Integer range 0..2 ** 14 - 1;

    -- Uniquely identifies a set of objects, within a particular world.  All
    -- versions of the object have the same pathname, class, file subclass
    -- (if appropriate), and Object_Index.

    type Simple_Object_Id is new Long_Integer range 0..2 ** 32 - 1;

    -- Uniquely identifies a set of objects, within this machine.  Beware that
    -- World_Number's and Object_Index's are subject to reuse.  Thus, there
    -- are dangling reference issues.  In some parts of the system (like
    -- Diana) there is (hopefully) sufficient obsolescence protection that the
    -- system will never interpret a dangling Object_Index.
    --   record
    --       Class : Class_Number;
    --               Filler.Filler_1;
    --       World : Machine.World_Number;
    --               Filler.Filler_2;
    --       Index : Object_Index;
    --   end record

    function Class (Id : Simple_Object_Id) return Class_Number;
    function World (Id : Simple_Object_Id) return Machine.World_Number;
    function Index (Io : Simple_Object_Id) return Object_Index;

    function To_Simple_Object_Id (Class : Class_Number;
                                  World : Machine.World_Number;
                                  Index : Object_Index)
                return Simple_Object_Id;

    -- Field extraction and aggregate operations.

    type Simple_Object_Id_Array is array (Natural range <>) of
                                       Simple_Object_Id;

    type Version_Number is new Long_Integer range 0..2 ** 24 - 1;
```

```
   -- Uniquely identifies a particular object, in the set of objects which
   -- have the same name.  Each member of the set must have a unique version
   -- number assigned.  Version numbers are assigned sequentially, in time,
   -- and are never reused.  Note, however, that objects can be updated
   -- without changing the version number.

   type Version_Id is private;
   subtype Object_Id is Version_Id;
   -- record
   --              : Filler.Filler_8;
   --     Machine  : Machine_Id;
   --     Instance : Unique_Id;
   --     Object   : Simple_Object_Id;
   --     Version  : Version_Number;
   -- end record; -- 17 bytes

   Nil_Version_Id : constant Version_Id;
   Nil_Object_Id  : constant Object_Id;

   -- The Machine field is used to make the object/version id unique across
   -- machines.

   -- The Instance field is used to match against the Instance field in the
   -- Segment_Descriptor of the referenced object and catches dangling
   -- references caused by reuse of World_Number's and Object_Index's.

   -- Note that a Version_Id does not indicate the generation of the version.
   -- To do that, one needs to drag along the Last_Update_Id field from the
   -- Segment_Descriptor for the version in question.

   -- The following conventions are used for the identification of job
   -- worlds: (Machine  => <the usual>,
   --          Instance => <that of the job world>,
   --          Object   => (Class => Class_Directory,
   --                       World => <job #>,
   --                       Index => <contant for world roots>)
   --          Version => Nil)

   -- The following conventions are used for the identification of job
   -- segment: (Machine  => <the usual>,
   --           Instance => <that of the segment>,
   --           Object   => (Class => Class_File,
   --                        World => <job #>,
   --                        Index => <1 + segment kind>)
   --           Version => <segment #>)

   function Instance (Id : Version_Id) return Unique_Id;
   function Extract_Machine (Id : Version_Id) return Machine_Id;
   function Object (Id : Version_Id) return Simple_Object_Id;
   function Version (Id : Version_Id) return Version_Number;
   function Class (Id : Version_Id) return Class_Number;
   function World (Id : Version_Id) return Machine.World_Number;
   function Index (Id : Version_Id) return Object_Index;

   function To_Version_Id (Instance : Unique_Id;
                           Class : Class_Number;
                           World : Machine.World_Number;
```

```
                            Index : Object_Index;
                            Version : Version_Number) return Version_Id;

    function To_Version_Id (Instance : Unique_Id;
                            The_Machine : Machine_Id;
                            Class : Class_Number;
                            World : Machine.World_Number;
                            Index : Object_Index;
                            Version : Version_Number) return Version_Id;
    function To_Version_Id (Instance : Unique_Id;
                            Object : Simple_Object_Id;
                            Version : Version_Number) return Version_Id;

    function To_Version_Id (Instance : Unique_Id;
                            Machine : Machine_Id;
                            Object : Simple_Object_Id;
                            Version : Version_Number) return Version_Id;

    function To_Version_Id (Object : Object_Id;
                            Version : Version_Number) return Version_Id;

    function To_Object_Id (Version : Version_Id) return Object_Id;

    type Version_Id_Array is array (Natural range <>) of Version_Id;

    type Version_Handle is private;
    subtype Object_Handle is Version_Handle;
    -- record
    --     Object : Version_Id;
    --     Universe : Version_Id;
    -- end record;

    Nil_Version_Handle : constant Version_Handle;
    Nil_Object_Handle  : constant Object_Handle;

    -- It is presumed that Version.Machine and Universe.Machine are identical.

    -- As the source of an operation: If Object_Handle.Object.Version is not
    -- Nil, then the specified version is used, and Object_Handle.Universe
    -- is ignored.  Otherwise, the Version_Number comes from the selected
    -- universe/world view; the "selected view" is specified by the .Universe
    -- component or job state when the .Universe component is Nil.

    -- As the target of an operation which does not create a new version, the
    -- rules are identical to those above.

    -- As the target of an operation which creates a new version,
    -- Object_Handle.Object.Version is ignored, and the new version is created
    -- in the selected universe/world view; the definition of "selected" is
    -- identical to that above.

    -- The subtype Object_Handle is used in contexts which simply ignore
    -- Object_Handle.Object.Version and Object_Handle.Universe, or in contexts
    -- which return Nil for those components. The subtype Version_Handle is
    -- used in the remaining contexts.

    function Object (Handle : Object_Handle) return Version_Id;
    function Universe (Handle : Object_Handle) return Version_Id;
```

```
   function To_Object_Handle (The_Object : Version_Id;
                              In_Universe : Version_Id) return Object_Handle;

   type Version_Handle_Array is
       array (Natural range <>) of Version_Handle;

   ---------------------
   -- Directory names --
   ---------------------

   subtype Bounded_String is Bounded.Variable_String;

   Max_Simple_Name_Length : constant := 63;
   subtype Simple_Name is Bounded_String (Max_Simple_Name_Length);
   subtype Simple_Name_String is String;
   subtype Padded_Simple_Name_String is String (1..63);
   Pad_Character          : constant Character := ' ';

   -- Names in the directory system have a maximum length, and are
   -- occasionally padded with blanks.  System objects often have a "'" at
   -- the beginning.  For example, the code segment for package "Foo" might
   -- be called "'Code", and be a child of "Foo".

   -- Every object has a name, even temporary heaps, system objects, etc.

   -- All versions of the same object have the same value for Name.

   Null_Name              : constant Simple_Name_String := "";

   -- A simple name is legal iff
   --          (1) Name'length <= max_simple_Name_Length,
   --      and (2) every character in the string is in the range ' ' .. '~',
   --      and (3) Name ">" Null_Name (1 .. Name'Length)
   -- Should just stick the BNF here.

   Max_Pathname_Length : constant := 511;
   subtype Pathname is Bounded_String (Max_Pathname_Length);
   subtype Pathname_String is String;

   -- A full pathname is a sequence of simple names separated by dots.
   -- Should stick the BNF here.

   subtype Simple_Wildcard is String;

   -- A simple name plus wildcard characters.
   -- Should stick the BNF here.  Special characters restricted to just
   -- "#" and "*".

   subtype Full_Wildcard is String;
   -- A full wildcard name spec.
   -- Should stick the BNF here.

   subtype Unique_Wildcard is Full_Wildcard;
   -- By convention, must resolve to a single object/version.

   subtype Naming_Context is Object_Handle;

   -- Can be used as the starting context for various operations which
```

```
   -- work on pathnames.

   ---------------------
   -- Version Control --
   ---------------------

   Max_Retained_Versions : constant := 15;

   subtype Version_Retention_Count is Natural
        range 0..Max_Retained_Versions;

   -- A World_View can keep track of up to Max_Retained_Versions of any
   -- particular object.


   type Version_Control_Policy is new Long_Integer range 0..2 ** 12 - 1;
   -- Applied to individual objects. See Object.Create.

   Non_Nil_Version_Control_Policy       : constant Version_Control_Policy :=
                                        1;
   -- Kludge bit to satisfy the Nil=0 convention.

   Allow_Explicit_Create_Version        : constant Version_Control_Policy :=
                                        2;
   -- When false, Object.Create will refuse to create new versions.

   Private_Version_On_First_Update      : constant Version_Control_Policy :=
                                        4;
   Private_Version_On_Every_Update      : constant Version_Control_Policy :=
                                        8;
   Implicit_Versions_Replace_Current    : constant Version_Control_Policy :=
                                        16;
   -- These 3 bits control when and how Object.Open (for update) creates
   -- implicit versions. See that spec for details.


   Freeze_Versions : constant Version_Control_Policy :=
                     32;
   -- Controls whether or not world.Freeze causes selected versions to be
   -- frozen.

   Prevent_Update_Without_View          : constant Version_Control_Policy :=
                                        64;
   -- When set, prevents Object.Open for update with non-nil version #.

   Shared_Version_Policy                : constant Version_Control_Policy :=
                                        Non_Nil_Version_Control_Policy;
   Standard_Differential_Version_Policy : constant Version_Control_Policy :=
                                        Non_Nil_Version_Control_Policy +
                                        Allow_Explicit_Create_Version +
                                        Private_Version_On_First_Update +
                                        Implicit_Versions_Replace_Current +
                                        Freeze_Versions +
                                        Prevent_Update_without_View;

   type Reservation_Bits is array (Object_Index range <>) of Boolean;

   -------------------------
```

```
    -- Info about Ada units --
    --------------------------

    type Compilation_State is
        (U0,
         Unparsed,          -- Not fully parsed
         Parsed,            -- Syntactically correct, not fully semanticized
         Installed,         -- Fully semantized
         Phase_1_Coded,     -- Unit attributed by the code generator
         Coded,             -- Code has been generated
         U6, U7);

    type Ada_Unit_State is
        record
            State                 : Compilation_State;

            Potentially_Obsolete : Boolean;
            -- An imported unit may have changed its meaning in a way that
            -- invalidates the current interpretation of the unit's semantics.


            Modified              : Boolean;
            -- Declarations have been edited which may have changed the meaning
            -- of one or more declarations exported by this unit.

        end record;

    type Dependency_Matrix is
        array (Object_Index range <>,
               Object_Index range <>) of Boolean;

    -- DM(J,K) is true iff unit J contains a semantic pointer to unit K.
    -- Additional information about the nature of the reference from J
    -- to K is stored in unit J.

    type Dependency_Vector is array (Object_Index range <>) of Boolean;

    -- Can contain either a row or column of the Dependency_Matrix.

    ----------------------
    -- Jobs and Logins --
    ----------------------

    subtype Job_Number is Machine.Job_Worlds;
    function Number (Tsk : Machine.Task_Id) return Job_Number;

    type Job_Id is new Long_Integer range 0..2 ** 58 - 1;
    --  record
    --      Instance : Unique_Id;
    --      Job      : Job_Number;
    --  end record;
    -- Job_Number uniquely identifies a job. Use of Job_Id protects one from
    -- the fact that Job_Number's are reused.

    function Instance (Id : Job_Id) return Unique_Id;
    function Number (Id : Job_Id) return Job_Number;

    function To_Job_Id (Instance : Unique_Id;
```

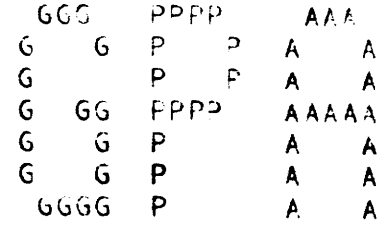
```
                    Job : Job_Number) return Job_Id;

    type Job_Kind is new Long_Integer range 0..2 ** 4 - 1;
    Terminated_Job : constant Job_Kind := 1;
    Attached_Job   : constant Job_Kind := 2;
    Detached_Job   : constant Job_Kind := 3;
    Server_Job     : constant Job_Kind := 4;
    Ce_Job         : constant Job_Kind := 5;
    Oe_Job         : constant Job_Kind := 6;

    type Job_Priority is new Long_Integer range 1..7;

    -- Priority 7 is "better" than priority 1. These values are used by the
    -- medium term scheduler. Not the same as Machine.Cpu_Priority_Level,
    -- which is used by the short term scheduler.

    type Job_State is new Long_Integer range 0..2 ** 4 - 1;
    Idle_Job    : constant Job_State := 1; -- no unblocked tasks
    Running_Job : constant Job_State := 2; -- has unblocked tasks
    Waiting_Job : constant Job_State := 3; -- withheld by mts
    Disable_Job : constant Job_State := 4; -- disabled by external agent
    Queued_Job  : constant Job_State := 5; -- detached and withheld by mts

    type Session_Number is new Long_Integer range 0..2 ** 8 - 1;

    type Session_Id is new Long_Integer range 0..2 ** 56 - 1;
    --  record
    --      Instance : Unique_Id;
    --      Session  : Session_Number;
    --  end record;
    -- The Session_Number uniquely identifies each "session".  Use of
    -- Session_Id protects one from the fact that Session_Number's are reused.

    function Instance (Id : Session_Id) return Unique_Id;
    function Number (Id : Session_Id) return Session_Number;

    function To_Session_Id (Instance : Unique_Id;
                            Session : Session_Number) return Session_Id;

    ---------------
    -- Execution --
    ---------------

    type Compatability_Index is new Long_Integer range 0..2 ** 16 - 1;
    type Compatability_Vector is
        array (Compatability_Index range <>) of Boolean;

    type Compatability_Key (Static_Length : Compatability_Index := 256) is
        record
            Vector         : Compatability_Vector (1..Static_Length);
            Dynamic_Length : Compatability_Index;
        end record;

    type Exception_Information is
        record
            Raise_Pc         : Machine.Address;
            Exception_Number : Long_Integer;
            Other_Info       : Long_Integer;
```

```
    end record;

  ----------------------------
  -- Users, Groups and ACL's --
  ----------------------------

  type User_Number is new Long_Integer range 0..2 ** 10 - 1;

  type User_Id is new Long_Integer range 0..2 ** 58 - 1;
  --  record
  --      Instance : Unique_Id;
  --      User     : User_Number;
  --  end record;
  -- The Instance and User_Number are computed from the directory object
  -- which corresponds to the user.

  function Instance (Id : User_Id) return Unique_Id;
  function User (Id : User_Id) return User_Number;

  function To_User_Id (Instance : Unique_Id;
                       User : User_Number) return User_Id;


  type Group_Number is new Long_Integer range 0..2 ** 12 - 1;
  Public : constant Group_Number := 1;

  type Group_Id is new Long_Integer range 0..2 ** 60 - 1;
  --   record
  --       Instance : Unique_id;
  --       Group    : Group_Number;
  --   end record;
  -- The Instance and Group_Number are computed from the directory object
  -- which corresponds to the group.

  function Instance (Id : Group_Id) return Unique_Id;
  function Group (Id : Group_Id) return Group_Number;

  function To_Group_Id (Instance : Unique_Id;
                        Group : Group_Number) return Group_Id;

  type Group_Map is array (Group_Number) of Boolean;

  type Acl_Entry_Kinds is new Long_Integer range 0..3;

  Prohibit_Access : constant Acl_Entry_Kinds := 0;
  Grant_Access    : constant Acl_Entry_Kinds := 1;
  Perform_Audit   : constant Acl_Entry_Kinds := 2;
  -- code 3 reserved for future use

  type Privilege_Bits is new Long_Integer range 0..2 ** 5 - 1;

  Create_Privilege  : constant Privilege_Bits := 2;
  Delete_Privilege  : constant Privilege_Bits := 1;
  Read_Privilege    : constant Privilege_Bits := 2;
  Write_Privilege   : constant Privilege_Bits := 4;
  Execute_Privilege : constant Privilege_Bits := 1;
  Owner_Privilege   : constant Privilege_Bits := 4;
  -- codes 8 & 16 reserved for future use
```

```
    type Acl_Entry is
        record
            Kind   : Acl_Entry_Kinds;

            -- Prohibit_Access causes the failure of an operation which
            -- requires the rights listed below and is requested by the
            -- identified Group, even if there is an entry later in the list
            -- which grants the desired access.  A user in the Privileged group
            -- with privileges enabled is never denied access, even if such an
            -- entry exists.

            -- Grant_Access allows an operation which requires the rights
            -- listed below and is requested by the identified Group.  There
            -- is an implicit entry giving the Privileged group all rights
            -- (this entry is not physically represented in the ACL).  If no
            -- entry grants the desired access, the operation fails.

            -- Perform_Audit means that the system will audit any opreration
            -- which requires the rights listed below and which is attempted
            -- and/or performed by the identified Group; the audit entry goes
            -- to the syster log.  The fact that an operation is audited does
            -- not effect whether the request meets the requirements specified
            -- by the remainder of the ACL. Auditing is performed even if the
            -- requester has Privileged access enabled.

            -- It is recommended (for performance reasons) that one place the
            -- entries which are most likely to match at the beginning of the
            -- list (subject to ordering by'Kind, of course).

            Group  : Group_Number;

            -- The group governed by this entry.  Nil terminates the list.
            -- Completely filling the array also terminates the list.

            Rights : Privilege_Bits;

            -- See visible part of the Access_Control spec, and the various
            -- OM operations for details.

        end record;

    type Access_Control_List is array (Natural range 1..6) of Acl_Entry;
    pragma Assert (Access_Control_List'Size = 120);

    Null_Acl_Entry : constant Acl_Entry :=
                     (Kind => Nil, Group => Nil, Rights => Nil);
    Null_Acl       : constant Access_Control_List :=
                     (others => Null_Acl_Entry);

    -- An element containing a Nil Group is considered to terminate the
    -- list.

    -- Note: moving an object from machine to machine or even to/from tape
    -- makes for interesting issues with respect to ACLs in the target copy.

    ------------
    -- Actions --
```

```
   ------------

   type Action_Number is new Long_Integer range 0..2 ** 12 - 1;

   -- Used as an index into the action manager's internal tables.
   -- OM subsystem reserves the right to change the size of this type.

   type Action_Id is new Long_Integer;
   -- Handle for an open action.
   --    record
   --        Instance : Unique_Id;
   --                 : Filler.Filler_4;
   --        Index    : Action_Number;
   --    end record

   function Boot (Id : Action_Id) return Boot_Number;
   function Instance (Id : Action_Id) return Unique_Id;
   function Index (Id : Action_Id) return Action_Number;

   function To_Action_Id (Instance : Unique_Id;
                          Index : Action_Number) return Action_Id;

   type Lock_Mode is new Long_Integer range 0..2 ** 4 - 1;

   Unsynchronized             : constant Lock_Mode := 1;
   Read_Object                : constant Lock_Mode := 2;
   Write_Object               : constant Lock_Mode := 3;
   Record_Read_Object         : constant Lock_Mode := 4;
   Record_Write_Object        : constant Lock_Mode := 5;
   Record_Read                : constant Lock_Mode := 6;
   Record_Write               : constant Lock_Mode := 7;
   Supersedeable_Read_Object  : constant Lock_Mode := 8;

   -- Documented in Action spec.

   subtype Record_Key is Long_Integer range 0..2 ** 32 - 1;

   -- When acquiring action locks, this value is used to identify a "record",
   -- within some object.

   --------------------
   -- Miscellaneous --
   --------------------

   type Relational_Op is (Lt, Le, Eq, Ne, Ge, Gt);

   type Format_Number is new Long_Integer range 0..2 ** 8 - 1;

   -- Used to tag various representations to facilitate detecting disk
   -- incompatable changes.

   type Segment_Position is new Long_Integer range 0..2 ** 32 - 1;

   -- Values of type Machine.Bit_Offset are used in Machine.Addresses to
   -- denote the physical position within a segment.  Values of type
   -- Segment_Position are used to denote the logical position within a
   -- segment. The two values differ by the amount of ucode & om overhead
   -- present at the beginning of the segment.
```

```
    First_Valid_Segment_Position : constant Segment_Position := 1;

    -- The Segment_Position type follows the usual rule for interpretation of
    -- Nil. The first bit of "user data" in a segmented heap occurs at
    -- Segment_Position'(1).

    type File_Organization_Kinds is new Integer range 0..15;

    -- Identifies the internal organization of a file, as defined by the file
    -- system. Hypothetical example: a file might be organized as raw bits (as
    -- in the Gamma system), or it might be organized as a sequence of records
    -- (which retain length information and support record level action
    -- semantics. In either case, one could imagine that a subclass of text
    -- would make sense; in the first organization, one might have raw
    -- characters in the file, with special characters used to denote line
    -- boundaries, etc; in the second organization, one might have lines
    -- stored in individual records.

    -------------
    -- Private --
    -------------

private
    type Net_Unique_Id is
        record
            Machine  : Machine_Id;
            Boot     : Boot_Number;
            Sequence : Unique_Sequence_Number;
        end record;

    Nil_Net_Unique_Id : constant Net_Unique_Id := (0, 0, 0);

    type Version_Id is
        record
            Fill     : Filler.Filler_8;
            Machine  : Machine_Id;
            Instance : Unique_Id;
            Object   : Simple_Object_Id;
            Version  : Version_Number;
        end record;

    Nil_Version_Id : constant Version_Id := (0, 0, 0, 0, 0);
    Nil_Object_Id  : constant Object_Id := Nil_Version_Id;

    type Version_Handle is
        record
            Version  : Version_Id;
            Universe : Version_Id;
        end record;

    Nil_Version_Handle : constant Version_Handle :=
                         (Nil_Version_Id, Nil_Version_Id);

    Nil_Object_Handle  : constant Object_Handle :=
                         (Nil_Version_Id, Nil_Version_Id);

end Om_Definitions;
```



User: GPA
Object: !DELTA_KK.REV1_0_0.UNITS.KK.KERNEL_SERVICES_1
Version: V(4)
Request: 1277

Date: April 24, 1986
Queued: 11:27:18 AM
Printed: 11:44:04 AM

```
with Sys;
with Machine;

package Kernel_Services_1 is
    -- pragma Subsystem (Kernel, Private_part => Closed);
    -- pragma Module_Name (4, ?);


    package Byte_String_Conversions is

        package System renames Sys;

        function To_String (B : System.Byte_String) return String;
        function To_Byte_String (S : String) return System.Byte_String;

    end Byte_String_Conversions;


    package Wait_Service is

        type Wait_State is private;
        Nil_Wait_State : constant Wait_State;

        function Register (Wait_State_Name : String) return Wait_State;

    -- Defines a new wait state.  If the given string is too long, or
    -- already defined, or the database is full, returns Nil_Wait_State.

    -- The following are predefined:
    --    Port_Wait, Port_Input_Wait, and Port_Output_Wait

        function Get_Wait_State (Wait_State_Name : String) return Wait_State;
        function Get_Wait_State_Name (State : Wait_State) return String;

    -- Converts from string name to Wait_State, and vice-versa.


        procedure Roust (The_Client : Machine.Task_Id;
                         State : Wait_State);

        procedure Un_Roust (State : Wait_State);

        Forever : constant Duration := Duration'Last;

        procedure Wait_For_Roust (State : Wait_State;
                                  Max_Wait : Duration := Forever);

    -- Assume the existance of some service S.  The clients of S, known as Ci,
    -- ask S to do work, and wish to be able to wait for the work to get done
    -- prior to proceeding.  We use S.task to denote the "worker" task of the
    -- service S.  We use S.skin to denote the procedures in the body of S
    -- which run on Ci's task, which is denoted Ci.task.

    -- Ci.task calls S, asking it to do some operation. S.skin should call
    -- Un_Roust; this serves the purpose of "throwing away" an outstanding
    -- roust that Ci.task did not see (these can happen because (a) the
    -- "roustedness" of a task is not initialized when the task is created,
    -- and (b) the task might have exited the skin of some service via delay
```

```
    -- expiration). S.skin then calls S.task, which attempts the operation.
    -- S.task then returns to S.skin a boolean indicating whether or not
    -- Ci.task should suspend.  If Ci.task is supposed to suspend, then S.skin
    -- calls Wait_For_Roust, on behalf of Ci.task; Ci.task becomes blocked
    -- inside wait_For_Roust.  At some later point, the operation completes,
    -- and Si.task invokes Roust; this will in turn cause Ci's call to
    -- Wait_For_Roust to return; S.skin may then want to call S.task to get
    -- the results of the operation.  (Its also possible for the Roust to
    -- happen prior to client invoking Wait_For_Roust.) It is also possible
    -- that the wait_For_Roust will return due to delay expiration, in which
    -- case S.skin will need to call S.task in order to get the results of the
    -- operation.

    -- The wait service has, in effect, a boolean semaphore per client.
    -- Therefore, when S.task plans to do a Roust on CI.task, S.skin should
    -- either guarantee that Ci.task has not left S.skin, or have some other
    -- guarantee that Ci.task is not attempting to wait on more than one event.

    -- Roust is a noop when the specified client no longer exists.

    -- Roust may be invoked by non-blocking tasks.

    -- The wait state parameter to the Roust and Jn_Roust procedures is used
    -- to ensure that services only roust clients that "are their skin".  That
    -- is, try to roust someone that is not in the specified state, and you
    -- will get constraint error.

    -- Notes for non-blocking (ie., priority 0) tasks: If the tcb of the
    -- rousted task is not in memory, the call to Roust will place a ghost call
    -- in the queue space of the roust server (a task internal to this
    -- package).  When the roust server accepts the call, it will execute the
    -- same roust, but at priority 1, thereby taking the necessary page fault.
    -- The roust server has a finite amount of queue space reserved.  In other
    -- words, up to N rousts of non-resident tcbs can be "outstanding".  Rousts
    -- remain outstanding because (a) priority 0 tasks hog the cpu keeping the
    -- roust server from running, or (b) the roust server is tied up in page
    -- fault servicing one of the rousts.  The current value for N is about
    -- 128.  Should the roust server's queue be full, the rousting priority 0
    -- task will attempt to create a new queue page; if the page cannot be
    -- created without calling the space manager, the system will crash.

    -- Currently, the only non-blocking service is the Port_Manager.  It is
    -- believed that the probability of crashing in this scenario is very low.

    private
        type Wait_State is new Integer range 0..2 ** 7 - 1;
        Nil_Wait_State : constant Wait_State := 0;

    end Wait_Service;


    package Trigger is
        -- Encapsulates the wait service protocal. Suitable for fairly
        -- trivial wait service applications.

        type Task_Count is new Natural range 0..2 ** 16 - 1;

        type Descriptor (Count : Task_Count) is private;
```

```
        procedure Initialize (Desc : in out Descriptor;
                              Wait_State : Wait_Service.Wait_State);

        -- Recommend COUNT to be prime.  You must INITIALIZE your descriptors.
        --  All the waiting tasks are assumed to be in the same wait state.

        procedure Pull (Desc : in out Descriptor);

        -- Rousts all the waiting tasks.


        generic
            with procedure Change (Tid : Machine.Task_Id; Is_Waiting : Boolean);

        procedure Wait_In_Skin
            (Wait_State : Wait_Service.Wait_State;
             Max_Wait : Duration;
             Non_Blocking_Service : Boolean := False);


        procedure Change (Desc : in out Descriptor;
                          Tid : Machine.Task_Id;
                          Is_Waiting : Boolean);

        -- Changes record of who is waiting for the trigger to be pulled.
        -- Feed this to WAIT_IN_SKIN generic.  Note that after calling change,
        -- you typically need to check if the "roust condition" holds, and if
        -- so pull the trigger.

        function Somebody_Is_Waiting (Desc : Descriptor) return Boolean;


    private

        type Waiter is
            record
                Tid : Machine.Task_Id;
            end record;

        type Waiter_Array is array (Task_Count range <>) of Waiter;

        type Descriptor (Count : Task_Count) is
            record
                Wait_State : Wait_Service.Wait_State;
                Waiters    : Waiter_Array (1..Count);
            end record;

    end Trigger;

end Kernel_Services_1;
```