IST-2001 34140

Programming Language Interfaces

Deliverable WP2-SI5-V3

Ian Broster, Alan Burns, Michael Gonzalez Harbour and Julio Medina

3 March 2005

Contents

1 Introduction

This deliverable concerns the Ada language interface to the computation model defined for FIRST. There are two aspects to defining an appropriate interface for FIRST:

1. how to incorporate the required functionality into the existing definition of Ada;
2. how to extend the provision of Ada so that it can more easily accommodate the needs of FIRST and similar approaches to flexible scheduling.

These two quite distinct activities are discussed separately in Sections 2 and 3 of this report.

2 FIRST interface to Existing Ada

Ada 95 (the current version of the Ada standard) gives no support for servers or application schedulers. Therefore an appropriate way to provide access to these facilities is required. The FSF API for Ada is similar to the C version and many of the functions and subprograms are have direct equivalents in the C interface. In this API, no pragmas are defined.

The API has been derived using the high-level requirements stated earlier in the project. For completeness the requirements for this API (as described in deliverable D-SI.1v2) are given in Appendix A.

2.1 Ada API

The final version of an Ada API has been developed following feedback from the previous iteration. The complete API is included in Appendix B of this deliverable in the form of an Ada package specification. It may be beneficial to read this specification in conjunction with the motivations for the C API, which is described in other FIRST deliverables D-OS.1v2 and D-OS.2v2.

The final version of the API resolves the remaining issues discussed in previous versions. In particular, the requirement for a budget during initialisation, background scheduling and service thread issues are resolved.

The key change to this version of the API is the introduction of modules. The CORE module interface (which is compulsory) is contained in the package 'fsf'. Other modules are implemented as child packages of fsf: fsf.hierarchical, fsf.shared_objects, fsf.spare_capacity, fsf.implementation_specific, fsf.distributed and fsf.distributed.spare_capacity.

2.2 Usage Scheme

In this section an outline of usage examples are presented, to illustrate the method of integration of the Ada API in the different contexts considered in FIRST.

Initialisation. There are two stages to initialisation. Firstly, the FSF itself must be initialised. This may well be done as part of the elaboration sequence as defined by the binder, or from the main program. Initialisation of FSF is achieved by calling the subprogram FSF.Init. After this, the FSF scheduling framework is active and the environment task will be running in background scheduling mode.

The second stage is the negotiation of the main contracts in the system. Two approaches for are considered, although an implementation need not necessarily support both ways. Firstly, a centralised approach is possible, where the main program negotiates each contract, binds processes to servers then blocks. This is ideal for closed systems, where there is no need for further contract negotiations.

The alternative approach is for the main program to leave contract negotiations to the individual processes. To ensure that this completes adaquately, the main process (or during elaboration) should set a contract for the service thread using Set_Service_task_Data. By default, the service thread executes in the background scheduling policy, which may be sufficient. However, a normal contract can be negotiated for the service task. The main program creates any processes and blocks. Each process then individually requests its own contract, which is negotiated using the budget of the special service thread. The negotiations are performed using the budget of the service thread, or in the background if the service thread finishes its budget during a negotiation.

Bounded workload. This kind of applications are characterised by being designed with conservative WCET, and bounded periods. Consider the following situations in which bounded workload servers are used.

Independent periodic tasks. Each task can be fit in an ad-hoc prepared server. If no useful usage of extra capacity can be done by these tasks, minimum and maximum values for budgets and periods can be made the same, which is done by simply not setting the reclamation parameters. Each job starts after returning from a call to the Schedule_Timed_Job procedure, the next time for activation is calculated as the last requested activation time plus the period authorised to the server, which is an output value of the same procedure.

Independent interrupt triggered tasks. When tasks are not strictly periodic but driven by generic external interrupts instead, but the minimum inter-arrival period and maximum per job computation time are known, the same model as with periodic tasks described above can be used. The basic difference is that at the beginning of each server's activation, just after returning from the Schedule_Timed_Job procedure, the job should block on the protected object corresponding to the expected external interrupt.

Event dispatcher With external interrupts, timing, particular software detected conditions or in general whichever event detection mechanism is centralized in a particular task, this task will act as a generic event dispatcher. In this case this task should be characterised and bound to a certain higher priority server. The tasks that are to be launched in response to every particular kind of event are in different servers and the budgets and periods for them are particularly designed. The mechanism used for the activation of these servers is the signalling of synchronization objects. In this case the activated tasks must

use the Schedule_Triggered_Job to wait for signalling and return the capacity not used in each job.

Unbounded workload. The unbounded workload applications do not usually have a clear minimum inter-arrival period though a certain number is assigned for the allocation of resources. Deadlines are not expected to be hard but mostly met. In this case the unbounded workload servers model is used. Basically it behaves like a sporadic server with the possible extension of budgets and/or reduction of periods according to the spare capacity sharing algorithm and the online unused reclaimed capacity returned by other servers in the system. No capacity is returned explicitly by these servers, since no job model is assigned. Tasks in this kind of server execute as much budget as they have according to the server rules imposed by the underlying scheduling mechanism. If a task blocks long enough, the capacity not used can be reclaimed in an automatic way by the system.

Enhanced background. Besides the usage of zero minimum budget and the quality and importance parameters for sharing spare capacity, some concrete possibilities for the accommodation of non real-time or relaxed constraints applications are:

Any-time algorithms. In this case bounded workload servers with continuous utilization granularity should be used. If the timing granularity is coarse enough, each processing cycle can be started by calculating or asking for the remaining budget in the present job (Get_Available_Capacity), if it is clearly not enough for a new cycle, the job can be ended and the remaining capacity returned (using Schedule_Timed_Job or Schedule_Triggered_Job).

N-version algorithms. Contracts for this kind of algorithms should ask for a range of budgets/periods combinations, probably with discrete utilization granularity. It is also advisable to use bounded workload servers, so that in each job the available capacity is used to select the adequate version of the algorithm.

Regular background. A round robin background scheduling scheme is available. To access this, a contract must be created with budget and period both set to 0 (contract negotiation may fail if there are shared objects). This server provides background scheduling; tasks requiring background scheduling may be bound to this server as required.

2.3 Implementation Advice

The FSF API presented here clearly contains significant features which are not trivial to implement. The existing implementation on MaRTE, which uses the GNAT compiler, provides a very POSIX-oriented interface to the underlying scheduling. Other styles of implementation are possible. Indeed, the ShARK reference implementation, which only supports the C API, uses a completely different underlying scheduling architecture.

The key features of FSF, which need to be implemented include: hierarchical scheduling, budget control and servers, synchronisation and application defined scheduling.

At the heart of the MaRTE OS reference implementation is a fixed priority scheduler. The preemption levels given in the contracts are translated into priorities by an implementation defined private function:

```
function Priority_Map (Plevel : in Preemption_Level)
    return System.Priority;
```

Further discussion of this and application based scheduling framework for Ada can be found in the 2004 Ada Europe paper, *Implementing an Application-Defined Scheduling Framework for Ada Tasking*[1] which is an output of the FIRST project. The operating systems and their interface to FSF will be fully described in separate documents OS1v3 and OS2v3.

Finally, as a reference for implementation, the private parts of the package specifications for fsf and fsf.distributed are presented in Appendix C.

2.4 Evaluation

Regarding quantitative evaluation, experimentation has been undertaken in terms of measuring overheads. The full report is presented in SI2v3, Scheduler Evaluation Report, where accurate measurements for API calls, context switches etc in MaRTE and SHaRK are given.

The initial results indicate that the FSF API behaves favourably. The times are increased slightly compared to (say) a native fixed priority scheduler because of the overhead of the advanced scheduling (servers, budgets, etc.), however the increase is slight.

Regarding usage evaluation, both case studies have used the Ada API to FSF and found it good to use. The number of lines of code required to use FSF was low, compared to the size of the application, and little effort was required to be able to use legacy applications designed for other operating systems. Feedback from the case studies was essential for the design of the final API. In particular, it was noted that the use of synchronisation objects and critical sections simplified the structure systems, making their design 'cleaner'.

3 Improving the Provisions of Ada

One of FIRST's major objectives is to attempt to influence the standards that are relevant to the support of flexible real-time systems. Within the context of this deliverable it is the definition of the Ada standard that is of concern. As an ISO standard, Ada's definition is reviewed every ten years. Current activity is concerned with possible amendments to the language that will take effect in 2005/6. These amendments are potentially wide ranging and include aspects of the type model and the introduction of interfaces. Members of the FIRST consortium have been concerned with the real-time annex and have focused on its scheduling provisions - with the objective of broadening its provisions. This work has produced:

- A collection of AIs (Ada Issues) that lay the foundation to the necessary amendments within the scope of scheduling. AIs are the mechanism by which language changes (amendments) are considered by the ISO authority responsible for Ada (this group is known as the ARG, which is itself a subgroup of ISO committee WG9). The AIs developed so far are as follows - the latest version of all of them are included are available on the ARG web site.
  - rewording of section D2.2 of the Ada Reference Manual (ARM) to allow extra scheduling to be added.
  - inclusion of non-preemption dispatching
  - dynamic priority ceilings for protected objects
  - execution-time budget control for individual tasks
  - group budget control for groups of tasks
  - timing events that allow code to be executed at defined times without the need to use a task with a delay statement
  - support for EDF dispatching and preemption-level locking
  - support for application-level scheduling - as used in FIRST.

  All of these proposals are making good progress with the exception of the application-level scheduling AI that has been deemed beyond the scope of the 2005 amendment.
- Three papers for the International Real-Time Ada Workshop that took place in September 2003, and which are now been published in Volume XXIII (4) December 2003 of Ada Letters:
  - *Task Attribute-Based Scheduling - Extending Ada's Support for Scheduling*[5]
  - *Application-Defined Scheduling in Ada* [7].
  - *Managing Multiple Execution-Time Timers from a Single Task*[6].
- A paper, presented at the 2003 Ada Europe Conference, arguing for the provisions of Ada to be expanded[3].
- A paper on EDF scheduling for 2004 Ada Europe Conference entitled *Supporting Deadlines and EDF Scheduling in Ada*[4].
- A paper on application level scheduling, referenced earlier, also for the 2004 Ada Europe Conference[1].
- A paper on the integration of the application-defined scheduling proposal with the evolution of the real-time AIs on EDF scheduling and execution time budgets, accepted for its presentation at the Ada-Europe 2005 conference[2].

Note that none of the AIs or the proposals in the papers go far enough in terms of giving direct support for servers. However the use of Timing Events and budget control does allow fixed priority servers to be programmed as part of the application. This has the advantage of allowing the particular semantics required by the application to be produced (rather than relying only of the provisions defined by the language). Although dynamic schemes such as EDF can be programmed using dynamic priorities a more effective and efficient scheme will come from direct support for EDF (as defined in the AI). The current proposal has the advantage that it will allow EDF and fixed priority (and other schemes) to coexist with the same application - this is clearly desirable for the form of flexibility envisaged and championed by FIRST.

The IRTAW meeting in September 2003 laid the foundation for much of the work on the AIs identified above. Member of FIRST participated fully in this workshop.

4 Conclusions

Much progress has been made in opening up the definition of Ada so that more flexible scheduling can be supported. However the extent to which these possibilities find their way into the language definition is as yet not decided. The development of an API for Ada has nevertheless illustrated the means by which this language can be used for programming flexible systems if the underlying run-time provides the necessary functionality.

References

[1] M. Aldea, J. Miranda, and M. González Harbour. Implementing an application-defined scheduling framework for Ada tasking. In *Reliable Software Technologies, Proceedings of the Ada Europe Conference*. Lecture Notes on Computer Science, Springer Verlag, (to appear), 2004.

[2] M. Aldea andJ. Miranda and M. González Harbour. Integrating application-defined scheduling with the new dispatching policies for ada tasks. In *Proceedings of the International Conference on reliable Software Technology, Ada-Europe,*, York, UK, June 2005. ACM.

[3] A. Burns, M. González Harbour, and A.J. Wellings. A round robin scheduling policy for Ada. In *Reliable Software Technologies, Proceedings of the Ada Europe Conference*, volume LNCS 2655, pages 334–343. Lecture Notes on Computer Science, Springer Verlag, 2003.

[4] A. Burns, M. González Harbour, and A.J. Wellings. Supporting deadlines and EDF scheduling in Ada. In *Reliable Software Technologies, Proceedings of the Ada Europe Conference*. Lecture Notes on Computer Science, Springer Verlag, (to appear), 2004.

[5] A. Burns and A.J. Wellings. Task attribute-based scheduling - extending Ada's support for scheduling. In T. Vardenega, editor, *Proceedings of the 12th International Real-Time Ada Workshop*, volume XXIII, pages 36–41. ACM Ada Letters, 2003.

[6] M. González Harbour and M.A. Rivas. Managing multiple execution-time timers form a single task. In T. Vardenega, editor, *Proceedings of the 12th International Real-Time Ada Workshop*, volume XXIII, pages 28–31. ACM Ada Letters, 2003.

[7] M.A. Rivas and M. González Harbour. Application-defined scheduling in Ada. In T. Vardenega, editor, *Proceedings of the 12th International Real-Time Ada Workshop*, volume XXIII, pages 42–51. ACM Ada Letters, 2003.

A A first draft of the Service Contract API

The API between the application or upper-level scheduler and the underlying scheduler will include at least the operations listed below. The contract parameters are specified as an opaque or private type, depending on the programming language facilities. Values of this type are only set through the interfaces defined by the following abstract operations, or additional ones that may be defined in the future:

- Initialize

  Input Data none

  Output Data Contract parameters object

  Description The operation initializes a contract parameters object setting it to the default values, and returns this object

- Set_Basic_Parameters

  Input Data Budget, Period, Workload (Bounded or Indeterminate), Contract parameters object

  Output Data Updated contract parameters object

  Description The operation updates the specified contract parameters object by setting its budget, period, and workload to the specified input parameters. (Note: the workload is a basic parameter because bounded tasks are triggered by the scheduler (see the Timed_Schedule_Next_Job operation, later), while indeterminate tasks are not; therefore, their programming model is quite different).

- Set_Timing_Requirements

  Input Data D=T (boolean), Deadline (must be null if D=T true), Budget Overrun Notification (may be null), Deadline Miss Notification (may be null), Contract parameters object

  Output Data Updated contract parameters object

  Description The operation updates the specified contract parameters object by setting its D=T and deadline parameters to the specified input parameters.

- Set_Reclamation_Parameters

  Input Data Granularity (continuous or discrete), Utilization set (null if continuous), Quality and Importance, Contract parameters object

  Output Data Updated contract parameters object

  Description The operation updates the specified contract parameters object by setting its granularity, utilization set, quality, and importance to the specified input parameters.

- Set_Synchronization_Parameters

  Input Data Preemption Level, Critical sections, Contract parameters object

  Output Data Updated contract parameters object

  Description The operation updates the specified contract parameters object by setting its preemption level and critical sections to the specified input parameters.

An abstract synchronization object is defined by the application. This object can be used by an application to wait for an event to arrive by invoking the Schedule_Triggered_Job operation. It can also be used to signal the event either causing a waiting server to wake up, or the event to be queued if no server is waiting for it. It has the following operations:

- Init

  Input Data None

  Output Data Initialized synchronization object

  Description This operation initializes a synchronization object managed by the scheduler.

- Signal

  Input Data Synchronization object

  Output Data None

  Description If one or more servers are waiting upon the specified synchronization object one of them is awakened; if not, the event is queued at the synchronization object.

- Negotiate_Contract

  Input data Contract parameters

  Output data Accepted (boolean), Server Id (Number)

  Description The operation negotiates a contract for a new server. If the on-line admission test is enabled it determines whether the contract can be admitted or not based on the current contracts established in the system. Then it creates the server and recalculates all necessary parameters for the contracts already present in the system. This is a potentially blocking operation; it returns when the system has either rejected the contract, or admitted it and made it effective.

- Cancel_Contract

  Input data Server_Id

  Output data None

Description The operation eliminates the specified server and recalculates all necessary parameters for the contracts remaining in the system. This is a potentially blocking operation; it returns when the system has made the changes effective.

- Renegotiate_Contract

  Input data Server Id, New Contract Parameters

  Output data Accepted (boolean)

  Description The operation renegotiates a contract for an existing server. If the on-line admission test is enabled it determines whether the contract can be admitted or not based on the current contracts established in the system. If it cannot be admitted, the old contract remains in effect. If it can be admitted, it recalculates all necessary parameters for the contracts already present in the system. This is a potentially blocking operation; it returns when the system has either rejected the new contract, or admitted it and made it effective.

- Request_Contract_Renegotiation

  Input data Server Id, New Contract Parameters, Notification (None, or signal number)

  Output data None

  Description The operation enqueues a renegotiate operation for an existing server, and returns immediately. The renegotiate operation is performed asynchronously, as soon as it is practical; meanwhile the system operation will continue normally. When the renegotiation is made, if the on-line admission test is enabled it determines whether the contract can be admitted or not based on the current contracts established in the system. If it cannot be admitted, the old contract remains in effect. If it can be admitted, it recalculates all necessary parameters for the contracts already present in the system. When the operation is completed, notification is made to the caller, if requested, via a signal. The status of the operation (in progress, admitted, rejected) can be checked with the `Renegotiation_Request_Status` operation.

- Change_Quality_And_Importance

  Input data New Quality, New Importance, Server Id

  Output data None

  Description The operation enqueues a request to change the quality and importance parameters of the specified server, and returns immediately. The change operation is performed as soon as it is practical; meanwhile the system operation will continue normally.

- Timed_Schedule_Next_Job

  Input data Server_Id, Absolute_time

  Output data Current Budget, Current Period, Deadline Missed (boolean), Budget Overran (boolean)

  Description This operation is invoked for bounded workload servers to indicate that a job has been completed (and that the scheduler may reassign the unused capacity of the current job to other servers), and also when the first job requires to be scheduled. The system will activate the job at the specified absolute time, and will then use the scheduling rules to determine when the job can run, at which time the call returns. Upon return, the system reports the current period and budget for the current job, whether the deadline of the previous job was missed or not, and whether the budget of the previous job was overrun or not.

- Event_Triggered_Schedule_Next_Job

  Input data Server_Id, Synchronization object,

  Output data Current Budget, Current Period, Deadline Missed (boolean), Budget Overran (boolean),

  Description This operation is invoked for bounded workload servers to indicate that a job has been completed (and that the scheduler may reassign the unused capacity of the current job to other servers), and also when the first job requires to be scheduled. If the specified synchronization object has events queued, one of them is dequeued; otherwise the server will wait upon the specified synchronization object until it is signalled. Then, the system will use the scheduling rules to determine when the job can run and the call will return at that time. Upon return, the system reports the current period and budget for the current job, whether the deadline of the previous job was missed or not, and whether the budget of the previous job was overrun or not.

  Overran (boolean)

- Available_Capacity

  Input data Server Id

  Output data Capacity, in percentage of utilization

  Description This operation returns the current spare capacity (in percentage of processor or network utilization), currently assigned to the importance level of the specified server.

- Total_Quality

  Input data Server Id

Output data Quality

Description This operation returns the sum of the quality parameters for all servers in the system of importance level equal to that of the specified server.

- Renegotiation_Request_Status

  Input data Server Id

  Output data In-progress, rejected, admitted

  Description The operation reports on the status of the last renegotiation operation enqueued for the specified server. It is callable even after notification of the completion of such operation, if requested.

- Is_Admission_Test_Enabled

  Input data None

  Output data Test Enabled Status (boolean)

  Description Returns true if the system is configured with the on-line admission test enabled, and false otherwise.

Operations to negotiate, cancel, and renegotiate multiple-server contracts will be available in the future (next phase of the project). For now, a multiple-server contract can be negotiated as a sequence of single-server contracts.

B Appendix B - Ada API

The following API has been developed within FIRST for the Ada programming language.

B.1 FSF Core Module

```
--=============================================================
--          FFFFFFFIII    RRRRR        SSTTTTTTT
--         FF             IIR    RR     SS
--        FF              IR           SS
--       FFFFFF           RRRR     SSSSST
--     FF          FI        RRR   SS
--   FF             II       RRR   SS
-- FF               IIIIIR      RS
--=============================================================
-- Basic FSF(FIRST Scheduling Framework) contract management

with Ada.Real_Time;
with Ada.Task_Identification;
with Ada.Unchecked_Conversion;
with System;
with POSIX_Signals;

package Fsf is

   --                    BASIC TYPES

   -- Types for the core module

   type Workload is (Bounded, Indeterminate, Overhead);

   -- Some constants used as default values

   Null_Deadline : constant Ada.Real_Time.Time_Span
     := Ada.Real_Time.Time_Span_Zero;

   Null_Signal    : constant POSIX_Signals.Signal
     := POSIX_Signals.Signal_Null;

   Null_Signal_Data : constant POSIX_Signals.Signal_Data;

   -- Types for the spare capacity module

   type Granularity is (Continuous, Discrete);

   -- Utilizations

   type Utilization_Value is record
```

```
   C : Ada.Real_Time.Time_Span;
   T : Ada.Real_Time.Time_Span;
end record;

type Utilization_Set is private;

Null_Utilization_Set : constant Utilization_Set;

Max_N_Utilization_Values : constant Natural := -- Not specified by the FSF.

type U_Index is range 0 .. Max_N_Utilization_Values;

subtype Positive_U_Index is U_Index range 1 .. U_Index'Last;

procedure Add
  (Set   : in out Utilization_Set;
   Value : in Utilization_Value);

function Size
  (Set : Utilization_Set)
  return U_Index;

function Element
  (Set   : Utilization_Set;
   Index : Positive_U_Index)
  return Utilization_Value;

-- Quality and inmportance

Max_N_Importance_Levels : constant Positive := -- Not specified by the FSF.

subtype Importance is Positive range 1 .. Max_N_Importance_Levels;

subtype Quality is Natural;

-- Types for the implementation specific module --

UINT32_MAX : constant := 2**32-1;
type Preemption_Level is range 1 .. UINT32_MAX;

-- Types for the shared objects module

-- Shared object identifier
Max_shared_obj_id_size : constant Natural := -- Not specified by the FSF.
type Shared_obj_id is new String (1 .. Max_shared_obj_id_size);

-- Shared object handle (opaque type)
type shared_obj_handle is private;

-- Critical sections
```

```
type Critical_Section_Data is record
   wcet : Ada.Real_Time.Time_Span;
   obj_handle : shared_obj_handle;
end record;

type Critical_Sections is private;

Null_Critical_Sections : constant Critical_Sections;

Max_N_Critical_Sections : constant Natural := -- Not specified by the FSF.

type CS_Index is range 0 .. Max_N_Critical_Sections;

subtype Positive_CS_Index is CS_Index range 1 .. CS_Index'Last;

procedure Add
  (Set   : in out Critical_Sections;
   Value : in Critical_Section_Data);

function Size
  (Set : Critical_Sections)
  return CS_Index;

function Element
  (Set   : Critical_Sections;
   Index : Positive_CS_Index)
  return Critical_Section_Data;

-- Types for the hierarchical module

-- Scheduling policies
type Sched_Policy is (FP, EDF, TABLE_DRIVEN, RR, NONE);

-- Scheduling parameters for the table-driven policy
-- list of target windows

type Target_Window_Data is record
   the_start : Ada.Real_Time.Time_Span;
   the_end : Ada.Real_Time.Time_Span;
   comp_time : Ada.Real_Time.Time_Span;
end record;

type table_driven_params is private;

Null_table_driven_params : constant table_driven_params;

Max_N_Target_Windows : constant Natural := -- Not specified by the FSF.

type TW_Index is range 0 .. Max_N_Target_Windows;
```

```
subtype Positive_TW_Index is TW_Index range 1 .. TW_Index'Last;

procedure Add
  (Set   : in out table_driven_params;
   Value : in Target_Window_Data);

function Size
  (Set : table_driven_params)
  return TW_Index;

function Element
  (Set   : table_driven_params;
   Index : Positive_TW_Index)
  return Target_Window_Data;


-- Scheduling policy and parameters
type Sched_Params (policy : Sched_Policy) is record
   case policy is
      when FP       => priority : Integer;
      when EDF      => deadline : Ada.Real_Time.Time_Span;
      when TABLE_DRIVEN => params : table_driven_params;
      when RR | NONE => null;
   end case;
end record;

-- Initialization information for a scheduling policy
type Sched_Init_Info (policy : Sched_Policy) is record
   case policy is
      when FP | EDF | NONE => null;
      when RR       => slice_duration : Ada.Real_Time.Time_Span;
      when TABLE_DRIVEN => schedule_duration : Ada.Real_Time.Time_Span;
   end case;
end record;


-- Types for the distributed services module

Max_N_Network_Ids : constant := -- Not specified by the FSF.

type Network_Id is new Natural range 0 .. Max_N_Network_Ids;

Null_Network_Id : constant Network_Id := 0; -- FSF_NULL_NETWORK_ID ;

Default_Network_Id : constant Network_Id := 1; -- FSF_DEFAULT_NETWORK_ID ;


--          BASIC CONTRACT PARAMETERS
--
```

```
type Contract_Parameters is private;


-- Initialize: The operation initializes and returns a contract
-- parameters object, setting it to the default values.
-- The default values are:
--  budget min and max are set to 0
--  period min and max are set to 0
--  the workload is unbounded (FSF_INDETERMINATE)
--  the server deadline is equal to the period
--  the budget and deadline overrun are not notified
--  the granularity is set to "continuous"
--  the quality and importance are set to 1
--  the scheduling policy is set to FSF_NONE
--  the network_id is set to null_network_id
--  the granted_capacity_flag is set to 0
function Initialize
  return Contract_Parameters;

-- Set_Contract_Basic_Parameters: The operation updates the
-- specified contract parameters object by setting its budget,
-- period, and workload to the specified input parameters.
-- (Note: the workload is a basic parameter because bounded
-- tasks are triggered by the scheduler (see the
-- Schedule_Next_Timed_Job operation, later), while
-- indeterminate tasks are not; therefore, their programming
-- model is quite different).
-- It may raise:
--  Bad_argument: if bugets or periods are zero or not
--    consistent
procedure Set_Contract_Basic_Parameters
  (Contract   : in out Contract_Parameters;
   Budget_Min : in Ada.Real_Time.Time_Span;
   Period_Max : in Ada.Real_Time.Time_Span;
   The_Workload : in Workload := Indeterminate);

-- Get_Contract_Basic_Parameters: This operation obtains
-- from the specified contract parameters object its
-- budget, period, and workload, and copies them to the
-- corresponding output parameters.
procedure Get_Contract_Basic_Parameters
  (Contract : in Contract_Parameters;
   Budget_Min : out Ada.Real_Time.Time_Span;
   Period_Max : out Ada.Real_Time.Time_Span;
   The_Workload : out Workload);

-- Set_Contract_Timing_Requirements: The operation updates the
-- specified contract parameters object. deadline must be
-- null_deadline if D_equals_t is 1 , Budget_overrun_signal
```

```
-- or Deadline_miss_signal may be null_signal (no notification)
-- or any posix signal. budget_overrun_signal_data and
-- deadline_miss_signal_data are the values to be delivered
-- with the signal (this version does not support deadline
-- values larger than the efective period). -- It may raise:
--  Bad_argument: if d_equals_t is 0 and ( deadline is
--    null_deadline or larger than the maximum period )
procedure Set_Contract_Timing_Requirements
  (Contract              : in out Contract_Parameters;
   D_Equals_T            : in Boolean := False;
   Deadline              : in Ada.Real_Time.Time_Span
     := Null_Deadline;
   Budget_Overrun_Signal : in POSIX_Signals.Signal
     := Null_Signal;
   Budget_Overrun_Signal_Data : in POSIX_Signals.Signal_Data
     := Null_Signal_Data;
   Deadline_Miss_Signal  : in POSIX_Signals.Signal
     := Null_Signal;
   Deadline_Miss_Signal_Data : in POSIX_Signals.Signal_Data
     := Null_Signal_Data);

-- Get_Contract_Timing_Requirements: The operation obtains the
-- corresponding output parameters from the specified contract
-- parameters object. If d_equals_t is 1 , the deadline will
-- not be updated
procedure Get_Contract_Timing_Requirements
  (Contract              : in Contract_Parameters;
   D_Equals_T            : out Boolean;
   Deadline              : out Ada.Real_Time.Time_Span;
   Budget_Overrun_Signal : out POSIX_Signals.Signal;
   Budget_Overrun_Signal_Data : out POSIX_Signals.Signal_Data;
   Deadline_Miss_Signal  : out POSIX_Signals.Signal;
   Deadline_Miss_Signal_Data : out POSIX_Signals.Signal_Data);


--
--              SYNCHRONIZATION OBJECTS
--

-- An abstract synchronization object is defined. This object
-- can be used by an application to wait for an event to arrive
-- by invoking the Schedule_Next_Event_Triggered_Job operation.
-- It can also be used to signal the event either causing a
-- waiting server to wake up, or the event to be queued if no
-- server is waiting for it. It is defined by the following
-- private type and has the following operations:


type Synch_Obj_Handle is private;
```

```
-- Create_Synch_Obj: This operation initializes a
-- synchronization object managed by the scheduler.
-- It may raise:
-- Internal_Error: if the operation fails because of
--   lack of memory
-- TOO_MANY_SYNCH_OBJS : if the number of synchronization
--   objects in the system has already exceeded the maximum
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is
--   not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not
--   running
function Create_Synch_Obj
  return Synch_Obj_Handle;


-- Signal_Synch_Obj: If one or more servers are waiting upon
-- the specified synchronization object one of them is
-- awakened; if not, the event is queued at the synchronization
-- object.
-- It may raise:
-- INVALID_SYNCH_OBJ_HANDLE if the handle is not valid
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--   scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not
--   running
-- TOO_MANY_EVENTS_IN_SYNCH_OBJ : if the number of events
--   stored in the synchronization object reaches the maximum
--   defined in the configuration parameter header file

procedure Signal_Synch_Obj
  (The_Synch_Obj_Handle : in Synch_Obj_Handle);


-- Destroy_Synch_Obj: This operation destroys the
-- synchronization object (created by a previous call to
-- Create_SynchObject) that is referenced by the
-- synchronization_object variable, after calling this
-- operation, the variable can not be used until it is
-- initialized again by a call to Create_SynchObject.
-- It may raise:
-- INVALID_SYNCH_OBJ_HANDLE if the handle is not valid
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--   scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not
--   running
procedure Destroy_Synch_Obj
  (The_Synch_Obj_Handle : in Synch_Obj_Handle);


--
```

```
--          SCHEDULING BOUNDED WORKLOADS
--

-- Schedule_Timed_Job: This operation is invoked by tasks
-- associated with bounded workload servers to indicate that
-- a job has been completed (and that the scheduler may
-- reassign the unused capacity of the current job to other
-- servers). It is also invoked when the first job of such
-- tasks has to be scheduled.
--
-- As an effect, the system will make the current server's '
-- budget zero for the remainder of the server's period, and '
-- will not replenish the budget until the specified absolute
-- time. At that time, all pending budget replenishments (if
-- any) are made effective. Once the server has a positive
-- budget and the scheduler schedules the calling task
-- again, the call returns and at that time, the system
-- reports the current period and budget for the current job,
-- whether the deadline of the previous job was missed or
-- not, and whether the budget of the previous job was
-- overrun or not.
--
-- In a system with hierarchical scheduling, since this call
-- makes the budget zero, the other tasks in the same server
-- are not run. As mentioned above, only when the call
-- finishes the budget may be replenished.
-- It may raise:
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not
--   running
-- INTERNAL_ERROR : erroneous binding or malfunction of the FSF
--   main scheduler
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--   scheduled under the FSF
-- NOT_BOUND : if the calling task does not have a valid
--   server bound to it
-- SERVER_WORKLOAD_NOT_COMPATIBLE: if the kind of workload of
--   the server is not FSF_BOUNDED
procedure Schedule_Timed_Job
  (Abs_Time        : in Ada.Real_Time.Time;
   Next_Budget     : out Ada.Real_Time.Time_Span;
   Next_Period     : out Ada.Real_Time.Time_Span;
   Was_Deadline_Missed : out Boolean;
   Was_Budget_Overran : out Boolean);


-- Schedule_Triggered_Job: This operation is invoked by tasks
-- associated with bounded workload servers to indicate that
-- a job has been completed (and that the scheduler may
-- reassign the unused capacity of the current job to other
-- servers). It is also invoked when the first job of such
```

```
-- tasks has to be scheduled. If the specified
-- synchronization object has events queued, one of them is
-- dequeued; otherwise the server will wait upon the
-- specified synchronization object, the server's budget will '
-- be made zero for the remainder of the server's period, and '
-- the implementation will not replenish the budget until the
-- specified synchronization object is signalled. At that
-- time, all pending budget replenishments (if any) are made
-- effective. Once the server has a positive budget and the
-- scheduler schedules the calling task again, the call
-- returns and at that time, the system reports the current
-- period and budget for the current job, whether the
-- deadline of the previous job was missed or not, and
-- whether the budget of the previous job was overrun or not.
--
-- In a system with hierarchical scheduling, since this call
-- makes the budget zero, the other tasks in the same server
-- are not run. As mentioned above, only when the call
-- finishes the budget may be replenished.
-- It may raise:
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- INTERNAL_ERROR : erroneous binding or malfunction of
--   the FSF main scheduler
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is
--   not scheduled under the FSF
-- NOT_BOUND : if the calling task does not have
--   a valid server bound to it
-- BAD_ARGUMENT : if the synch_handle given is not
--   valid
-- SERVER_WORKLOAD_NOT_COMPATIBLE: if the kind of
--   workload of the server is not FSF_BOUNDED
procedure Schedule_Triggered_Job
 (Synch_Handle    : in Synch_Obj_Handle;
  Next_Budget     : out Ada.Real_Time.Time_Span;
  Next_Period     : out Ada.Real_Time.Time_Span;
  Was_Deadline_Missed : out Boolean;
  Was_Budget_Overran : out Boolean);


-- Timed_Schedule_Triggered_Job: This call is the same as
-- fsf_schedule_triggered_job, but with an absolute timeout.
-- The timed_out argument, indicates whether the function
-- returned because of a timeout or not
-- It may raise:
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- INTERNAL_ERROR : erroneous binding or malfunction of
--   the FSF main scheduler
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is
```

```
--    not scheduled under the FSF
-- NOT_BOUND : if the calling task does not have
--    a valid server bound to it
-- BAD_ARGUMENT : if the synch_handle given is not
--    valid
-- SERVER_WORKLOAD_NOT_COMPATIBLE: if the kind of
--    workload of the server is not FSF_BOUNDED
procedure Timed_Schedule_Triggered_Job
  (Synch_Handle    : in Synch_Obj_Handle;
   Abs_timeout     : in Ada.Real_Time.Time;
   Timed_out       : out Boolean;
   Next_Budget     : out Ada.Real_Time.Time_Span;
   Next_Period     : out Ada.Real_Time.Time_Span;
   Was_Deadline_Missed : out Boolean;
   Was_Budget_Overran : out Boolean);


--
--            CONTRACT NEGOTIATION OPERATIONS
--
-- The following functions are used to create servers for a
-- contract parameters specification and also to assign tasks
-- to servers

type Server_Id is private;

-- Negotiate_Contract: The operation negotiates a contract for
-- a new server. If the on-line admission test is enabled it
-- determines whether the contract can be admitted or not based
-- on the current contracts established in the system. Then it
-- creates the server and recalculates all necessary parameters
-- for the contracts already present in the system. This is a
-- potentially blocking operation; it returns when the system
-- has either rejected the contract, or admitted it and made it
-- effective. It returns 1 in the accepted boolean argument
-- and places the server identification number in the
-- the_server output parameter if accepted, or 0 in the
-- accepted output argument if rejected. No task is bound to
-- the newly created server, which will be idle until a task is
-- bound to it. This operation can only be executed by tasks
-- that are already bound to an active server and therefore are
-- being scheduled by the fsf scheduler.
-- It may raise:
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not
--    running
-- INTERNAL_ERROR : erroneous binding or malfunction of the FSF
--    main scheduler
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--    scheduled under the FSF
procedure Negotiate_Contract
```

```
   (Contract : in Contract_Parameters;
    Accepted : out Boolean;
    The_Server : out Server_Id);


   -- Negotiate_Contract_For_Myself: This operation negotiates a
   -- contract for a new server, and binds the calling task to it.
   -- If the contract is accepted it places the server
   -- identification number in the the_server output parameter, or
   -- an exception is raised if rejected.
   --
   -- Implementation dependent issue: In order to allow the usage
   -- of application defined schedulers, the calling task must not
   -- have the SCHED_APP scheduling policy and at the same time be
   -- attached to an application scheduler different than the fsf
   -- scheduler; in such case, an error is returned. After a
   -- successful call the calling task will have the SCHED_APP
   -- scheduling policy and will be attached to the fsf scheduler.
   -- It may raise:
   --  UNKNOWN_APPSCHEDULED_TASK : if the calling task is
   --    attached to an application defined scheduler different
   --    than the fsf scheduler
   --  CONTRACT_REJECTED : if the contract is rejected.
   --  SERVER_WORKLOAD_NOT_COMPATIBLE: if the kind of workload in
   --    the contract is FSF_OVERHEAD
   procedure Negotiate_Contract_For_Myself
     (Contract : in Contract_Parameters;
      The_Server : out Server_Id);


   -- Bind_Task_To_Server: This operation associates a task with a
   -- server, which means that it starts consuming the budget of
   -- the server and is executed according to the contract
   -- established for that server. If the task is already bound to
   -- another server, it is effectively unbound from it and bound
   -- to the specified one.
   -- It fails if the policy of the server is different than
   -- FSF_NONE, or if there is already a task bound to this
   -- server
   --
   -- Implementation dependent issue: In order to allow the usage of
   -- application defined schedulers, the given task must be already
   -- attached to the fsf scheduler, or be the calling task.
   --
   -- It may raise:
   --  INTERNAL_ERROR : erroneous binding or malfunction of the FSF
   --    main scheduler
   --  UNKNOWN_APPSCHEDULED_TASK : if the task is attached to
   --    an application defined scheduler different than the fsf
   --    scheduler
```

```
-- BAD_ARGUMENT : if the server value does not complain with
--   the expected format or valid range or the given task
--   does not exist
-- NOT_CONTRACTED_SERVER : if the referenced server is not
--   valid
-- SERVER_WORKLOAD_NOT_COMPATIBLE: if the kind of workload of
--   the server is FSF_OVERHEAD
-- ALREADY_BOUND : if the given server has a task already
--   bound
procedure Bind_Task_To_Server
  (The_Server : in Server_Id;
   The_Task : in Ada.Task_Identification.Task_Id);


-- Unbind_Task_From_Server: This operation unbinds a task from
-- a server. Since tasks with no server associated are not
-- allow to execute, they remain in a dormant state until they
-- are either eliminated or bound again.
--
-- If the task is inside a critical section the effects of
-- this call are deferred until the critical section is ended
--
-- Implementation dependent issue: in the implementation with
-- an application scheduler, the task is still attached to the
-- fsf scheduler, but suspended.
-- It may raise:
-- INTERNAL_ERROR : erroneous binding or
--   malfunction of the FSF main scheduler
-- BAD_ARGUMENT : if the given task does not
--   exist
-- NOT_SCHEDULED_TASK : if the given task is
--   not scheduled under the FSF
-- UNKNOWN_APPSCHEDULED_TASK : if the task is
--   attached to an application defined scheduler different
--   than the fsf scheduler
-- NOT_BOUND : if the given task does not
--   have a valid server bound to it
procedure Unbind_Task_From_Server
  (The_Task : in Ada.Task_Identification.Task_Id);


-- Get_Server: This operation returns the Id of the server
-- associated with the specified task.
-- It may raise:
-- NOT_SCHEDULED_TASK: if the given task is not under
--   the control of the scheduling framework
-- NOT_BOUND : if the given task does not have a valid
--   server bound to it
-- BAD_ARGUMENT : if the given task does not exist
function Get_Server
```

```
   (The_Task : in Ada.Task_Identification.Task_Id)
   return Server_Id;


-- Get_Contract: This operation returns the contract parameters
-- currently associated with the specified server.
-- It may raise:
--  BAD_ARGUMENT :if the given server id is incorrect
--  NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--    scheduled under the FSF
--  INVALID_SCHEDULER_REPLY : the scheduler is wrong or not running
--  NOT_CONTRACTED_SERVER : if the server of the calling task
--    has been cancelled or it is not valid
function Get_Contract
  (The_Server : in Server_Id)
  return Contract_Parameters;


-- Cancel_Contract: The operation eliminates the specified
-- server and recalculates all necessary parameters for the
-- contracts remaining in the system. This is a potentially
-- blocking operation; it returns when the system has made the
-- changes effective.
-- It may raise:
--  BAD_ARGUMENT :if the given server id is incorrect
--  NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--    scheduled under the FSF
--  INVALID_SCHEDULER_REPLY : the scheduler is wrong or not running
--  NOT_CONTRACTED_SERVER : if the server of the calling task
--    has been cancelled or it is not valid
procedure Cancel_Contract
  (The_Server : in Server_Id);


-- Renegotiate_Contract: The operation renegotiates a contract
-- for an existing server. If the on-line admission test is
-- enabled it determines whether the contract can be admitted
-- or not based on the current contracts established in the
-- system. If it cannot be admitted, the old contract remains
-- in effect and returns 0 . If it can be admitted, it
-- recalculates all necessary parameters for the contracts
-- already present in the system and returns 1 . This is a
-- potentially blocking operation; it returns when the system
-- has either rejected the new contract, or admitted it and
-- made it effective.
-- It may raise:
--  BAD_ARGUMENT :if the given server id is incorrect
--  NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--    scheduled under the FSF
--  INVALID_SCHEDULER_REPLY : the scheduler is wrong or not running
```

```
-- NOT_CONTRACTED_SERVER : if the server of the calling task
--   has been cancelled or it is not valid
function Renegotiate_Contract
  (New_Contract : in Contract_Parameters;
   The_Server : in Server_Id) return Boolean;


-- Request_Contract_Renegotiation: The operation enqueues a
-- renegotiate operation for an existing server, and returns
-- immediately. The renegotiate operation is performed
-- asynchronously, as soon as it is practical; meanwhile the
-- system operation will continue normally. When the
-- renegotiation is made, if the on-line admission test is
-- enabled it determines whether the contract can be admitted
-- or not based on the current contracts established in the
-- system. If it cannot be admitted, the old contract remains
-- in effect. If it can be admitted, it recalculates all
-- necessary parameters for the contracts already present in
-- the system. When the operation is completed, notification is
-- made to the caller, if requested, via a signal. The status
-- of the operation (in progress, admitted, rejected) can be
-- checked with the Get_Renegotiation_Status operation. The
-- argument notification can be null_signal (no notification),
-- or any posix signal; and in this case notification_data is
-- to be sent with the signal.
-- It may raise:
-- BAD_ARGUMENT :if the given server id is incorrect or
--   sig_notify is neither Null_Signal nor a valid POSIX signal
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--   scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not running
-- NOT_CONTRACTED_SERVER : if the server of the calling task
--   has been cancelled or it is not valid
procedure Request_Contract_Renegotiation
  (New_Contract : in Contract_Parameters;
   The_Server     : in Server_Id;
   Notification   : in POSIX_Signals.Signal := Null_Signal;
   Notification_Data : in POSIX_Signals.Signal_Data :=
     Null_Signal_Data);


type Renegotiation_Status is
  (In_Progress, Rejected, Admitted, Not_Requested);

-- Get_Renegotiation_Status: The operation reports on the
-- status of the last renegotiation operation enqueued for
-- the specified server. It is callable even after
-- notification of the completion of such operation, if
-- requested. If the fsf_request_contract_renegotiation
-- operation has not ever been called for the given server
```

```
-- the status returned is NOT_REQUESTED
-- It may raise:
-- BAD_ARGUMENT :if the given server id is incorrect
-- NOT_SCHEDULED_CALLING_TASK : if the calling task is not
--   scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--   task has been cancelled or it is not valid
function Get_Renegotiation_Status
  (The_Server : in Server_Id)
  return Renegotiation_Status;


--
--       OBTAINING INFORMATION FROM THE SCHEDULER
--

type Capacity is range 0 .. UINT32_MAX;

-- Is_Admission_Test_Enabled: It Returns 1 if the system
-- is configured with the on-line admission test enabled,
-- or 0 otherwise.
function Is_Admission_Test_Enabled
  return Boolean;


-- Get_Cpu_Time:This function returns the current execution
-- time spent by the task or tasks that are bound to the
-- specified server.
-- It may raise:
-- BAD_ARGUMENT : if the value of the server argument
--   is incorrect
-- NOT_SCHEDULED_CALLING_TASK : if the calling task
--   is not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--   task has been cancelled or it is not valid
function Get_Cpu_Time
  (The_Server : in Server_Id)
  return Ada.Real_Time.Time_Span;


-- Get_Remaining_Budget: This function returns the
-- remaining execution-time budget associated with the
-- specified server in the present cycle.
-- It may raise:
-- BAD_ARGUMENT : if the value of the server argument
--   is incorrect
```

```
-- NOT_SCHEDULED_CALLING_TASK : if the calling task
--   is not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--   task has been cancelled or it is not valid
function Get_Remainig_Budget
  (The_Server : in Server_Id)
  return Ada.Real_Time.Time_Span;


-- Get_Budget_And_Period: This function returns in the
-- output arguments cycle_budget and cycle_period, the
-- execution-time budget and the period respectively
-- associated with the specified server.
-- It may raise:
-- BAD_ARGUMENT : if the value of the server argument
--   is incorrect
-- NOT_SCHEDULED_CALLING_TASK : if the calling task
--   is not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--   task has been cancelled or it is not valid
procedure Get_Budget_And_Period
  (The_Server : in Server_Id;
   Cycle_Budget : out Ada.Real_Time.Time_Span;
   Cycle_Period : out Ada.Real_Time.Time_Span);


--
--          SERVICE TASK TUNING
--

-- Set_Service_Task_Data: This function allows the
-- application to change the period and budget of the
-- service task that makes the negotiations. Increasing the
-- utilization reserved for this task makes the
-- negotiations faster, but introduces additional load in
-- the system that may decrease the capacity available for
-- the servers. For this call, the system will make a
-- schedulability analysis to determine if the new
-- situation is acceptable or not and this condition is
-- returned by the function. If the new service task data
-- is accepted, the system will reassign budgets and
-- periods to the servers according to the new bandwidth
-- available, in the same way as it does for a regular
-- contract negotiation.
--
-- When its budget is exhausted, the service task may run
```

```
-- in the background
--
-- The service task starts with a default budget and
-- period that are configurable
--
-- Implementation dependency: in the fixed priority
-- implementtaion of fsf, the default priority is lower
-- than the priority of any server, but higher than the
-- background. According to the implementation-dependent
-- module the priority is adjustable by means of a function
-- that changes its preemption level
-- It may raise:
-- BAD_ARGUMENT : if the value of the budget value is
--   greater than the value of the period
-- NOT_SCHEDULED_CALLING_TASK : if the calling task
--   is not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--   task has been cancelled or it is not valid
function Set_Service_Task_Data
  (budget : Ada.Real_Time.Time_Span;
   period : Ada.Real_Time.Time_Span)
  return Boolean;


-- Get_Service_Task_Data: this function returns in the
-- output variables budget and period, respectively, the
-- current budget and period of the service task.
-- It may raise:
-- NOT_SCHEDULED_CALLING_TASK : if the calling task
--   is not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--   not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--   task has been cancelled or it is not valid
procedure Get_Service_Task_Data
  (budget : out Ada.Real_Time.Time_Span;
   period : out Ada.Real_Time.Time_Span);


--
--          BACKGROUND MANAGEMENT
--

-- A round-robin background scheduling policy is available
-- for those tasks that do not have real-time requirements.
-- Because some of these tasks may require sharing
-- information with other tasks run by regular servers,
-- special background contracts may be created for
```

```
-- specifying the synchronization requirements.
--
-- The way of specifying a background contract is by
-- setting budget_min = period_max = 0. Negotiation may
-- fail if the contract uses shared_objects. If the
-- contract has no shared_objects the returned server id
-- represents the background and may be used to bind more
-- than one task. If the contract has shared objects a
-- server is created to keep track of them, but the
-- associated tasks are executed in the background,
-- together with the other background tasks


--
--          CHANGE OF MODE: GROUPS OF CONTRACTS
--

-- Necesary Data types

type Contract_Set is private;

Null_Contract_Set : constant Contract_Set;

Max_N_Contract_Values : constant Natural := -- Not specified by the FSF.

type C_Index is range 0 .. Max_N_Contract_Values;

subtype Positive_C_Index is C_Index range 1 .. C_Index'Last;

procedure Add
  (Set   : in out Contract_Set;
   Value : in Contract_Parameters);

function Size
  (Set : Contract_Set)
  return C_Index;

function Element
  (Set   : Contract_Set;
   Index : Positive_C_Index)
  return Contract_Parameters;


type Server_Set is private;

Null_Server_Set : constant Server_Set;

Max_N_Server_Values : constant Natural := -- Not specified by the FSF.

type S_Index is range 0 .. Max_N_Server_Values;
```

```
subtype Positive_S_Index is S_Index range 1 .. S_Index'Last;

procedure Add
  (Set   : in out Server_Set;
   Value : in Server_Id);

function Size
  (Set : Server_Set)
  return S_Index;

function Element
  (Set   : Server_Set;
   Index : Positive_S_Index)
  return Server_Id;

-- Negotiate_group: This operation analizes the
-- schedulability of the context that results from
-- negotiating the contracts specified in the contracts_up
-- set and cancelling the contracts referenced by the
-- servers_down set. If the overall negotiation is
-- successful, a new server will be created for each of the
-- elements of the contracts_up group, the servers in the
-- servers_down set will be cancelled, the set of new
-- server ids will be returned in the output variable
-- servers_up, and the Accepted output variable will return
-- 1 . Otherwise, the Accepted output variable will
-- return 0 , and no other effect will take place.
-- Observe that in order to be able to receive the returned
-- arguments, the server of the calling task should not be
-- in the servers_down list.
-- It may raise:
-- INTERNAL_ERROR : erroneous binding or malfunction of
--    the FSF main scheduler
-- BAD_ARGUMENT : if both the contracts_up and
--    servers_down arguments are respectively
--    Null_Contract_set and Null_Server_set or any of them
--    has erroneous size or its elements are corrupted or
--    not in the valid range respectively
-- NOT_SCHEDULED_CALLING_TASK : if the calling task
--    is not scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or
--    not running
-- NOT_CONTRACTED_SERVER : if the server of the calling
--    task has been cancelled or it is not valid
procedure Negotiate_group
  (contracts_up : in Contract_Set;
   servers_down : in Server_Set;
   servers_up   : out Server_Set;
   accepted     : out Boolean);
```

```
--
--          INITIALIZATION SERVICES
--

-- Init: No fsf function can be called before this
-- operation. After calling Init, the calling task (the
-- main program) will be executing in the background. Then,
-- it can do the negotiations, create the tasks and, if
-- needed, activate them via some user-specified
-- synchronization mechanism. It may also create a contract
-- for itself. The second time this function is called it
-- fails.
--
-- Implementation issue: When using the application defined
-- scheduling mechanism the tasks that are to be
-- incorporated in the application defined scheduler must
-- be in the ready state. Because of this, two basic
-- approachs are envisioned to initialize the system, in a
-- centralized approach the main program will call Init,
-- negotiate all the contracts (maybe with the
-- negotiate_group operation), create all the tasks
-- dynamically with a priority set staticallly to a value
-- lower than the background, and then do the corresponding
-- bind operations. If a non- centralized initialization
-- scheme is used, each task may try to initialize the fsf
-- subsystem by calling Init, and then negotiate its own
-- contract (with negotiate_contract_for_myself).
-- It may raise:
-- SYSTEM_ALREADY_INITIALIZED : if the function has
--    already been called before
-- It may also raise any other exception due to errors
-- that may be encountered by the underlying operating
-- system primitives required to perform the FSF system
-- start up
procedure Init;


Too_Many_Tasks            : exception;
Bad_Argument              : exception;
Invalid_Synch_Obj_Handle  : exception;
No_Renegotiation_Requested : exception;
Contract_Rejected         : exception;
Not_Scheduled_Calling_Task : exception;
Not_Bound                 : exception;
Unknown_Appscheduled_Task : exception;
Not_Contracted_Server     : exception;
Not_Scheduled_Task        : exception;
Too_Many_Service_Jobs     : exception;
```

```
   Too_Many_Synch_Objs     : exception;
   Too_Many_Servers_In_Synch_Obj : exception;
   Too_Many_Events_In_Synch_Obj : exception;
   Internal_Error          : exception;
   Too_Many_Servers        : exception;
   Invalid_Scheduler_Reply : exception;
   Too_Many_Pending_Replenishments : exception;
   System_Already_Initailized : exception;
   Shared_Obj_Already_Initialized : exception;
   Shared_Obj_Not_Initialized : exception;
   Sched_Policy_Not_Compatible : exception;
   Workload_Not_Compatible : exception;
   Already_Bound           : exception;
   Wrong_Network           : exception;
   Too_Large               : exception;
   Buffer_Full             : exception;
   No_Space                : exception;
   No_Messages             : exception;
   Module_Not_Supported    : exception;
   System_Not_Initialized  : exception;
   Too_Many_Shared_Objs    : exception;


   -- functions that encapsulate the configuration information
   -- about the modules that are supported (set in the private part)
   function is_Hiearchical_Module_Supported
     return Boolean;
   function is_Distributed_Module_Supported
     return Boolean;
   function is_Distributed_Spare_Capacity_Module_Supported
     return Boolean;
   function is_Spare_Capacity_Module_Supported
     return Boolean;
   function is_Implementation_Specific_Module_Supported
     return Boolean;
   function is_Dynamic_Reclaiming_Module_Supported
     return Boolean;
   function is_Shared_Objects_Module_Supported
     return Boolean;

private

   -- Not specified by the FSF.

end Fsf;
```

B.2 FSF Spare Capacity Module

```
--=============================================================
--          FFFFFFFII     RRRRR         SSTTTTTTT
--        FF             IIR    RR      SS
--       FF               IR           SS
--      FFFFFF             RRRR       SSSSST
--    FF          FI         RRR   SS
--   FF            II       RRR   SS
-- FF               IIIII R      RS
--=============================================================

-- Basic FSF(FIRST Scheduling Framework)
-- spare capacity sharing functionality


package Fsf.Spare_Capacity is

   -- Set_Contract_Reclamation_Parameters: The operation updates the
   -- specified contract parameters object by setting its maximum usable
   -- budget, minimum period, granularity, utilization set, quality, and
   -- importance to the specified input parameters.
   -- It may raise:
   -- Bad_argument: if
   --   (budget_max value is grater than period_max or smaller than budget_min) or
   --   (period_min is smaller than budget_min or larger than period_max) or
   --   (granularity is neither FSF_CONTINUOUS nor FSF_DISCRETE) or
   --   (granularity is FSF_CONTINUOUS and
   --    utilization_set is not FSF_NULL_UTILIZATION_SET)    or
   --   (granularity is FSF_DISCRETE and
   --    utilization_set is FSF_NULL_UTILIZATION_SET)        or
   --   (utilization_set is not FSF_NULL_UTILIZATION_SET and
   --    (size of utilization_set less than 2 or greater
   --     than FSF_MAX_N_UTILIZATION_VALUES) )               or
   --   (the utilization_set elements are not in increasing utilization order) or
   --   (the first utilization value in the utilization_set does not match
   --    the pair (budget_min, period_max) of the contract)  or
   --   (the last utilization value in the utilization_set does not match
   --    the pair (budget_max, period_min) of the contract)
   -- Constraint_error: if
   --   (quality < 0)                                       or
   --   (importance is less than 1 or greater than FSF_N_IMPORTANCE_LEVELS)
   procedure Set_Contract_Reclamation_Parameters
     (Contract        : in out Fsf.Contract_Parameters;
      Budget_Max      : in Ada.Real_Time.Time_Span;
      Period_Min      : in Ada.Real_Time.Time_Span;
      The_Granularity : in Fsf.Granularity := Fsf.Continuous;
      The_Utilization_Set : in Fsf.Utilization_Set
                          := Fsf.Null_Utilization_Set;
      The_Quality     : in Fsf.Quality  := Fsf.Quality'First;
```

```
   The_Importance : in Fsf.Importance := Fsf.Importance'First);


-- Get_Contract_Reclamation_Parameters: The operation obtains from
-- the specified contract parameters object its maximum budget,
-- minimum period,granularity, utilization set, quality, and importance.
-- All the utilization_values of the utilization_set are copied, if the
-- array given to the_Utilization_Set is not unconstrained,
-- the constraint_error exception could be raised.
procedure Get_Contract_Reclamation_Parameters
  (Contract        : in Fsf.Contract_Parameters;
   Budget_Max      : out Ada.Real_Time.Time_Span;
   Period_Min      : out Ada.Real_Time.Time_Span;
   The_Granularity : out Fsf.Granularity;
   The_Utilization_Set : out Fsf.Utilization_Set;
   The_Quality     : out Fsf.Quality;
   The_Importance : out Fsf.Importance);


-- Request_Change_Quality_And_Importance: The operation enqueues a
-- request to change the quality and importance parameters of the
-- specified server, and returns immediately. The change operation is
-- performed as soon as it is practical; meanwhile the system operation
-- will continue normally.
-- It may raise:
-- BAD_ARGUMENT : if the value of the server argument is not in range
-- Constraint_error: if (quality < 0) or (importance is less than 1 or
--   greater than FSF_N_IMPORTANCE_LEVELS)
-- NOT_SCHEDULED_CALLING_THREAD : if the calling thread is not
--   scheduled under the FSF scheduler
-- INVALID_SCHEDULER_REPLY: if the scheduler is wrong or not running
-- NOT_CONTRACTED_SERVER: if the server has been cancelled or it
--   is not valid
procedure Request_Change_Quality_And_Importance
  (The_Server  : in Fsf.Server_Id;
   New_Importance : in Fsf.Importance;
   New_Quality : in Fsf.Quality);


-- Get_Total_Quality: This operation calculates and returns the sum of
-- the quality parameters for all servers in the system of importance
-- level equal to that of the specified server.
-- It may raise:
-- BAD_ARGUMENT : if the value of the server argument is not in range.
-- NOT_SCHEDULED_CALLING_THREAD : if the calling thread is not
--   scheduled under the FSF
-- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not running.
-- NOT_CONTRACTED_SERVER : if the server has been cancelled or it
--   is not valid
 function Get_Total_Quality
```

```
    (The_Server : in Fsf.Server_Id)
    return Natural;

  -- get_available_capacity: This operation returns the spare capacity
  -- currently assigned to the importance level of the specified server.
  -- The capacity represents the processor or network utilization and it is
  -- the number returned divided by Fsf.UINT32_MAX (2**32 - 1).
  -- It may raise:
  -- BAD_ARGUMENT : if the value of the server argument is not in range
  -- NOT_SCHEDULED_CALLING_THREAD : if the calling thread is not
  --   scheduled under the FSF
  -- INVALID_SCHEDULER_REPLY : the scheduler is wrong or not running
  -- NOT_CONTRACTED_SERVER : if the server has been cancelled or it
  --   is not valid
  function Get_Available_Capacity
    (The_Server : in Fsf.Server_Id)
    return Fsf.Capacity;

end Fsf.Spare_Capacity;
```

B.3 FSF Hierarchical Scheduling Module

```
--=============================================================
--          FFFFFFFII     RRRRR         SSTTTTTTT
--        FF             IIR    RR      SS
--       FF               IR           SS
--      FFFFFF             RRRR      SSSSST
--     FF          FI        RRR   SS
--   FF             II       RRR   SS
-- FF                IIIIIR      RS
--=============================================================

-- FSF(FIRST Scheduling Framework)
-- hierarchical scheduling management

with Ada.Task_Identification;

package Fsf.Hierarchical is

   -- init_local_scheduler: This call has the following effects:
   -- FP: none
   -- EDF: none
   -- TABLE_DRIVEN :
   --   Records the schedule duration, and starts the
   --   schedule at the time of the call. After the
   --   schedule duration has elapsed, the schedule in
   --   the table is repeated.
   -- It may raise:
   -- BAD_ARGUMENT: if the value of the server argument is not valid
   -- NOT_SCHEDULED_CALLING_THREAD: if the calling task is not
   --   scheduled under the FSF
   -- INVALID_SCHEDULER_REPLY: if the scheduler is wrong or not running
   -- NOT_CONTRACTED_SERVER: if the server of the calling task
   --   has been cancelled or it is not valid
  procedure init_local_scheduler (
    server : in Fsf.server_id;
    info   : in Fsf.sched_init_info);


   -- ////////////////////////////////////////////////
   -- //                CONTRACT PARAMETERS
   -- ////////////////////////////////////////////////

   -- set_contract_scheduling_policy: The operation updates the
   -- specified contract parameters object by setting its scheduling
   -- policy to the specified input parameter. The default policy is
   -- FSF_NONE, which means that only one task may be bound to the
   -- server.
  procedure Set_Contract_scheduling_policy
    (Contract       : in out Fsf.Contract_Parameters;
```

```
   The_sched_policy : in Fsf.Sched_policy := Fsf.None);

-- get_contract_scheduling_policy: This operation returns the
-- scheduling policy of the specified contract parameters object
function Get_Contract_scheduling_policy
  (Contract : in Fsf.Contract_Parameters)
  return Fsf.Sched_policy;


-- bind_local_thread_to_server: This operation associates a task
-- with a server, which must have a policy different than FSF_NONE. The
-- task's local scheduling parameters are set to the value stored in
-- sched_params, which must be compatible with the server's scheduling
-- policy. After the call the task starts consuming the server's
-- budget and is executed according to the contract established
-- for that server and to its scheduling policy. If the task was
-- already bound to another server, it is effectively unbound from
-- it and bound to the specified one.
--
-- Implementation dependent issues:
-- - In order to allow the usage of application defined schedulers,
--   the given task must not be attached to an application scheduler
--   different than the fsf scheduler.
-- - In the present version of the application defined scheduling
--   mechanism in MaRTE OS, the creation of the local tasks and the
--   bind must be performed dynamically from a higher priority task,
--   so that the task is in the ready state at the time of assigning
--   it to the FSF application defined scheduler.
-- It may raise:
-- BAD_ARGUMENT : if the the_server argument does not complain with
--   the expected format or valid range, or the given task does not
--   exist.
-- SCHED_POLICY_NOT_COMPATIBLE : if the scheduling policy
--   in sched_params is not compatible to the server's one.
-- INTERNAL_ERROR : erroneous binding or malfunction of the FSF
--   main scheduler
-- UNKNOWN_APPSCHEDULED_TASK : if the task is attached to
--   an application defined scheduler different than the fsf scheduler
-- NOT_CONTRACTED_SERVER : if the referenced server is not valid
-- SERVER_WORKLOAD_NOT_COMPATIBLE: if the kind of workload
--   of the server is FSF_OVERHEAD
procedure Bind_Local_Task_To_Server
  (the_Server : in Fsf.Server_Id;
   the_Task   : in Ada.Task_Identification.Task_Id;
   sched_params : in Fsf.Sched_Params);


-- set_local_task_sched_parameters: this function changes the
-- local scheduling parameters of the task to the value given
-- by sched_params. This value must be compatible to the
```

```
   -- scheduling policy of the server to which the task is bound.
   -- It may raise:
   -- BAD_ARGUMENT : if the given task does not exist.
   -- SCHED_POLICY_NOT_COMPATIBLE : if the task is already bound
   --   and the scheduling policy in sched_params is not compatible
   --   to the one of the task's server.
   -- NOT_SCHEDULED_TASK : if the given task is not scheduled
   --   under the FSF
   -- INTERNAL_ERROR : erroneous binding or malfunction of the FSF
   --   main scheduler
   -- UNKNOWN_APPSCHEDULED_TASK : if the task is attached to an
   --   application defined scheduler different than the fsf scheduler
   -- NOT_CONTRACTED_SERVER : if the task is bound and its server
   --   is not valid
   procedure set_local_task_sched_parameters
     (the_Task   : in Ada.Task_Identification.Task_Id;
      sched_params : in Fsf.Sched_Params);


   -- get_local_task_sched_parameters: this function returns the
   -- local scheduling parameters of the specified task
   -- It may raise:
   -- BAD_ARGUMENT: if the task does not exist
   -- NOT_SCHEDULED_THREAD: if the given thread is not scheduled
   --   under the FSF
   function get_local_task_sched_parameters
     (the_Task   : in Ada.Task_Identification.Task_Id)
     return Fsf.Sched_Params;


end Fsf.Hierarchical;
```

B.4 FSF Shared Objects Module

```
--=============================================================
--          FFFFFFIII     RRRRR         SSTTTTTTT
--         FF              IIR     RR      SS
--        FF               IR            SS
--       FFFFFF             RRRR      SSSSST
--     FF          FI         RRR   SS
--   FF             II       RRR   SS
-- FF                IIIIIR      RS
--=============================================================

-- Basic FSF(FIRST Scheduling Framework)
-- shared objects functionality


package Fsf.Shared_Objects is

   --       SHARED OBJECTS & OPERATIONS MANAGEMENT
   --

   -- init_shared_object: Initialization of shared objects. If the
   -- object identified by obj_id does not yet exist it is created, a
   -- handle to the object is returned. If the object already exists,
   -- the function fails.
   -- It may raise:
   --  Bad_argument: if obj_id is the null string
   --  SHARED_OBJ_ALREADY_INITIALIZED: if the object identified
   --    by obj_id already exists.
  function init_shared_object
    (obj_id : Fsf.shared_obj_id)
    return Fsf.shared_obj_handle;


   -- get_shared_object_handle: getting the handle of shared objects.
   -- If the object already exists a handle to the object is returned.
   -- Otherwise, an error code is returned by the function.
   -- It may raise:
   --  Bad_argument: if obj_id is the null string
   --  SHARED_OBJ_NOT_INITIALIZED: if the object identified
   --    by obj_id does not exists.
   --  NOT_SCHEDULED_CALLING_THREAD : if the calling task is not
   --    scheduled under the FSF
   --  INVALID_SCHEDULER_REPLY : the scheduler is wrong or not
   --    running
   --  NOT_CONTRACTED_SERVER : if the server of the calling task
   --    has been cancelled or it is not valid
  function get_shared_object_handle
    (obj_id : Fsf.shared_obj_id)
    return Fsf.shared_obj_handle;
```

```
  --                    CONTRACT PARAMETERS
  --

  -- set_contract_synchronization_parameters: The operation updates
  -- the specified contract parameters object by setting its critical
  -- sections to the specified input parameter.
  procedure Set_Contract_Synchronization_Parameters
    (Contract           : in out Fsf.Contract_Parameters;
     The_Critical_Sections : in Fsf.Critical_Sections
                          := Fsf.Null_Critical_Sections);


  -- fsf_get_contract_synchronization_parameters: The operation obtains
  -- from the specified contract parameters object its critical sections,
  -- and returns them.
  function get_contract_synchronization_parameters
    (Contract           : Fsf.Contract_Parameters)
    return Fsf.Critical_Sections;

end Fsf.Shared_Objects;
```

B.5 FSF Implementation Defined Module

```
--=============================================================
--          FFFFFFFII    RRRRR        SSTTTTTTT
--        FF            IIR    RR     SS
--       FF              IR           SS
--      FFFFFF           RRRR      SSSSST
--     FF         FI       RRR   SS
--   FF           II       RRR   SS
-- FF              IIIII R      RS
--=============================================================

-- Basic FSF(FIRST Scheduling Framework)
-- implementation specific functionality

package Fsf.Implementation_Specific is

  -- set_contract_preemption_level: The operation updates the
  -- specified contract parameters object by setting its preemption level
  -- to the specified input parameter.
  procedure Set_Contract_Preemption_Level
    (Contract         : in out Fsf.Contract_Parameters;
     The_Preemption_Level : in Fsf.Preemption_Level
                          := Fsf.Preemption_Level'First); --'


  -- get_contract_preemption_level: The operation obtains from the
  -- specified contract parameters object its preemption level and
  -- returns it.
  function Get_Contract_Preemption_Level
    (Contract : Fsf.Contract_Parameters)
    return Fsf.Preemption_Level;


  -- set_service_task_preemption_level: this function sets the
  -- preemption level of the service task to the specified value.
  -- The initial preemption level is a configurable parameter. This
  -- value is stored in a temporary variable and it is used the next
  -- time the service thread data is updated with the
  -- fsf_set_service_thread_data function

  procedure set_service_task_preemption_level
    (The_Preemption_Level : in Fsf.Preemption_Level);

  -- get_service_task_preemption_level: this function returns the
  -- current preemption level of the service task
  function get_service_task_preemption_level
    return Fsf.Preemption_Level;

  -- set_shared_obj_preemption_level: The operation updates the
```

```
  -- specified shared object by setting its preemption level
  -- to the specified input parameter.
  -- OBSERVATION: if this value is changed being any contract that
  -- uses the resource already accepted, the system's behavior and
  -- particularly the acceptance tests correctness are not garantee
  -- and probably wrong.
  procedure set_shared_obj_preemption_level
    (obj_handle     : in Fsf.Shared_obj_handle;
     preemption_level : in Fsf.Preemption_level
                        := Fsf.Preemption_Level'Last);

  -- fsf_get_shared_obj_preemption_level: The operation obtains from the
  -- specified shared object its preemption level and copies
  -- it to the place pointed to by the specified input parameter.
  function get_shared_obj_preemption_level
    (obj_handle : in Fsf.shared_obj_handle)
    return Fsf.Preemption_level;

end Fsf.Implementation_Specific;
```

B.6 FSF Distributed Module

```
--=============================================================
--            FFFFFFFII     RRRRR          SSTTTTTTT
--          FF              IIR     RR      SS
--         FF               IR            SS
--       FFFFFF             RRRR      SSSSST
--     FF          FI          RRR   SS
--   FF             II        RRR   SS
-- FF                IIIII R       RS
--=============================================================

-- FSF(FIRST Scheduling Framework)
-- distributed services functionality
with Ada.Streams;
package Fsf.Distributed is

   -- This operation should be called before any other of
   -- the distributed module. If it is not called, all of the
   -- other operations will raise System_Not_Initialized.
   -- It may raise:
   --   System_Already_Initailized: if it is not the first time it
   --   is called.
   procedure Init;

   -- This operation identifies a contract as a
   -- bandwidth reservation on the network identified
   -- by the The_network_id input parameter. Then the
   -- contract negotiation is performed using the
   -- conventional negotiation functions, including
   -- the possibility of grouping the contract with
   -- others. If the The_network_id given is
   -- Fsf.Null_Network_id, the contract is considered
   -- not to reserve bandwidth in a network but to
   -- operate as any other regular contract. It
   -- It may raise:
   --   Bad_argument: if the network id is not valid.
   procedure Set_Contract_Network_id
     (Contract     : in out Fsf.Contract_Parameters;
      The_Network_id : in Fsf.Network_Id := Fsf.Default_Network_id);

   -- This operation returns the network identification
   -- corresponding to the contract parameters object
   -- identified by contract. If the contract is a regular
   -- one and therefore it has not a network_id set, it
   -- returns the Fsf.NULL_NETWORK_ID constant instead.

   function Get_Contract_Network_id
     (Contract     : in Fsf.Contract_Parameters)
     return Fsf.Network_Id;
```

```
-- Transmission services:

-- private types for fsf endpoints
type Send_endpoint is private;
type Receive_endpoint is private;

-- The node_address type specifies the node address
-- in a communication-protocol-independent way. The actual
-- address is obtained via a configuration dependent mapping
-- function
subtype Node_address is Natural;

-- The port type specifies the information that is
-- necessary to get in contact with the thread in the
-- receiving node, in a protocol-independent way.
-- The actual port number is obtained via a configuration
-- dependent mapping function
subtype Port is Natural;

-- This operation creates a unidirectional input
-- data endpoint through which, after the
-- corresponding binding, it is possible to send
-- data. network_id identifies the network to use,
-- receiver specifies the communication protocol
-- dependent information that is necessary to
-- address the receiving node, and port specifies
-- the communication protocol dependent information
-- that is necessary to get in contact with the
-- desired destination.
-- It may raise:
--  Bad_Argument : if the_network_id or the_port are
--    not valid.
procedure
  create_send_endpoint
  (the_network_id : in Fsf.network_id := Fsf.Default_Network_id;
   receiver       : in Node_address;
   the_port       : in Port;
   endpoint       : out Send_endpoint);

-- This operation eliminates any resources reserved
-- for the referenced endpoint. If the endpoint is
-- bound to a network server, it is unbound from it
-- and can not be further used to invoke send
-- operations on it.
-- It may raise:
--  Bad_Argument : if endpoint is not valid.
procedure
  destroy_send_endpoint
```

```
  (endpoint : in out Send_endpoint);

-- This operation returns in the variables the_network_id,
-- receiver, and port, the corresponding
-- parameters used in the creation of the given
-- send endpoint.
-- It may raise:
--   BAD_ARGUMENT: if endpoint is not valid.
procedure
  get_send_endpoint_parameters
  (endpoint     : in Send_endpoint;
   the_network_id : out Fsf.Network_id;
   receiver     : out Node_address;
   the_port     : out Port);

-- This operation associates a send endpoint with a
-- server, which means that messages sent through
-- that endpoint will consume the server's reserved
-- bandwidth and its packets will be sent according
-- to the contract established for that server. If
-- the endpoint is already bound to another server,
-- it is effectively unbound from it and bound to
-- the specified one.
-- It may raise:
-- BAD_ARGUMENT: if endpoint or server are not valid.
-- ALREADY_BOUND: if the server is currently bound
--   to any other send endpoint.
-- WRONG_NETWORK: if the server's network id is not
--   the same as the one in the endpoint
procedure
  bind_endpoint_to_server
  (server  : in Fsf.server_id;
   endpoint : in out Send_endpoint);

-- This operation unbinds a send endpoint from a
-- server. Endpoints with no server associated
-- cannot be used to send data, and they stay in
-- that state until they are either eliminated or
-- bound again.
-- It may raise:
-- BAD_ARGUMENT: if endpoint is not valid.
-- NOT_BOUND: if the endpoint has no server
--   bound.
procedure
  unbind_endpoint_from_server
  (endpoint : in out Send_endpoint);

-- This operation returns the id of the server that
-- is bound to the specified send endpoint.
-- It may raise:
```

```
-- BAD_ARGUMENT: if endpoint is not valid.
function
  get_endpoint_server
  (endpoint : in Send_endpoint)
  return Fsf.Server_Id;

-- This operation sends the message stored in msg
-- through the given endpoint. The operation is non-blocking and
-- returns immediately. An internal fsf service will schedule the
-- sending of messages and implement the communications sporadic server
-- corresponding to the network server bound to the given endpoint.
-- Messages sent through the same endpoint are received in the same
-- order in which they were sent.
-- It may raise:
-- BAD_ARGUMENT: if endpoint or msg are not valid.
-- NOT_BOUND: if the endpoint has not a valid
--   server bound.
-- TOO_LARGE: if the message is too large for the
--   network provider (protocol) used
-- BUFFER_FULL: if the sending queue is full
procedure
  send
  (endpoint : in Send_endpoint;
   msg      : in Ada.Streams.Stream_Element_Array);


-- This operation creates a receive endpoint with all the information
-- that is necessary to receive information from the specified network
-- and port.
-- It may raise:
-- BAD_ARGUMENT: if the port or the network id are not valid.
procedure
  create_receive_endpoint
  (the_network_id : in Fsf.network_id := Fsf.Default_Network_id;
   the_port       : in Port;
   endpoint       : out Receive_endpoint);

-- This operation eliminates any resources reserved
-- for the referenced endpoint. it
-- can not be further used to invoke receive
-- operations on it.
-- It may raise:
-- Bad_Argument : if endpoint is not valid.
procedure
  destroy_receive_endpoint
  (endpoint : in out Receive_endpoint);

-- This operation returns in the variables the_network_id,
-- and port, the corresponding
-- parameters used in the creation of the given
```

```
-- receive endpoint.
-- It may raise:
--    BAD_ARGUMENT: if endpoint is not valid.
procedure
  get_receive_endpoint_parameters
  (endpoint     : in Receive_endpoint;
   the_network_id : out Fsf.Network_Id;
   the_port     : out Port);

-- If there are no messages available in the specified receive endpoint
-- this operation blocks the calling thread waiting for a message to be
-- received. When a message is available, if its size is less than or
-- equal to the length of the stream Data, the function stores it in the
-- stream and puts the number of bytes received in the variable Last.
-- Messages arriving at a receiver buffer that is full will be silently
-- discarded. The application is responsible of reading the receive
-- endpoints with appropriate regularity, or of using a sequence number
-- or some other mechanism to detect any lost messages.
-- It may raise:
-- BAD_ARGUMENT: if the endpoint is not valid.
-- NO_SPACE: if the stream message is too small for the message
--    received (in which case the message is lost).
-- NOT_BOUND: if the endpoint has not a valid
--    server bound.
 procedure
   receive
   (endpoint : in Receive_endpoint;
    Data    : out Ada.Streams.Stream_Element_Array;
    Last    : out Ada.Streams.Stream_Element_Offset);

-- This operation is the same as receive, except
-- that if there are no messages available in the
-- specified receive endpoint at the time of the call
-- the operation raises NO_MESSAGES
 procedure
  try_receive
  (endpoint : in Receive_endpoint;
   Data    : out Ada.Streams.Stream_Element_Array;
   Last    : out Ada.Streams.Stream_Element_Offset);

-- Getting configuration dependent distributed information section

-- This operation returns the transmission time that it takes
-- to send a packet through the network designated by
-- network_id, when there is no contention, but including any
-- network overheads. It is used to calculate the minimum and
-- maximum budgets used in the preparation of network
-- contracts. The effective network utilization budget
-- (usually called bandwidth) is always assiged as a number
-- of packets per time unit, so the time used in the
```

```
   -- negotiation of contracts will be internally transformed
   -- into the corresponding necessary number of packets.
   -- It may raise:
   --  BAD_ARGUMENT: if network_id is not valid.
   function packet_tx_time
     (the_network_id : in Fsf.network_id := Fsf.Default_Network_id)
     return Ada.Real_Time.Time_Span;

   -- This operation returns the maximum number of bytes that
   -- can be sent in a packet through the network designated by
   -- network_id. It is usually a configuration value and it
   -- helps the application user to calculate the number of
   -- packets it will need to reserve for the transmision of its
   -- messages and consequently prepare the corresponding
   -- contract.
   -- It may raise:
   --  BAD_ARGUMENT: if network_id is not valid.
   function packet_size
     (the_network_id : in Fsf.network_id := Fsf.Default_Network_id)
     return Natural;

   -- This operation is used to obtain the maximum number of
   -- packets of which a message can be formed, for the
   -- specified network. A message is defined as the piece of
   -- information used in a send operation. Since the value
   -- returned by this operation is measured in packet units,
   -- the effective size can be calculated multiplying this
   -- value by the size of a packet. When the value returned
   -- by this operation is larger than 1 it means the
   -- implementation will make the partition of messages into
   -- packets and its recomposition at the receiving node.
   -- It may raise:
   --  BAD_ARGUMENT: if network_id is not valid.
   function max_message_size
     (the_network_id : in Fsf.network_id := Fsf.Default_Network_id)
     return Natural;


private

   -- Not specified by the FSF.

end Fsf.Distributed;
```

B.7 FSF Distributed Spare Capacity

```
--=============================================================
--          FFFFFFFIII    RRRRR        SSTTTTTTT
--        FF              IIR    RR     SS
--       FF               IR           SS
```

```
--       FFFFFF              RRRR       SSSSST
--      FF          FI         RRR    SS
--    FF             II       RRR    SS
-- FF                IIIIIR       RS
--=============================================================
```

```
-- FSF(FIRST Scheduling Framework)
-- distributed spare capacity functionality

package Fsf.Distributed.Spare_Capacity is


   -- This operation sets the granted capacity flag in the contract
   -- parameters object pointed to by contract to the boolean specified in
   -- granted_capacity_flag. This flag indicates to the scheduler that
   -- once the negotiation of the respective contract is finished, the
   -- first values for the budget and period given to the corresponding
   -- server must not be changed due to the negotiation or renegotiation
   -- of any other contract in the system. The period can change, though,
   -- if a renegotiation or a change of quality and importance is
   -- requested for the corresponding server.
   -- It may raise:
   --  Bad_argument: if the network id is not valid.

   procedure Set_Contract_granted_capacity_flag
     (Contract            : in out Fsf.Contract_Parameters;
      granted_capacity_flag : in boolean := true);


   -- This operation returns the value of the granted_capacity_flag
   -- in the contract parameters object designated by contract.

   function Get_Contract_granted_capacity_flag
     (Contract            : in Fsf.Contract_Parameters)
     return boolean;


   -- This operation is used to return spare capacity that was assigned to
   -- a server but that cannot be used due to restrictions in other
   -- servers of a distributed transaction. This operation changes the
   -- cycle period and budget of the given server to the values given in
   -- new_period and new_budget, respectively.
   -- It may raise:
   -- Bad_argument: if the server does not have the granted_capacity_flag
   --    set, if the new period is less than the current one, or if the
   --    new budget is larger than the current one. It also raises this
   --    exception if the new period is greater than the maximum period
   --    currently specified in the contract associated to the server or,
   --    similarly, if the new budget is smaller than the minimum budget
   --    in the contract. It also raises this exception if the
```

```
   --    granularity is discrete and the new period and budget do not
   --    match any of the period−budget pairs in the utilization set of
   --    the server.

  procedure set_server_capacity
     (the_server : in Fsf.server_id;
      New_Period : in Ada.Real_Time.Time_Span;
      New_Budget : in Ada.Real_Time.Time_Span);


end Fsf.Distributed.Spare_Capacity;
```

C Private Part Reference

The private specifications of fsf and fsf.distributed are presented here as a reference implementation.

C.1 FSF private part

```
private

   -- Maximum number of accepted contracts (servers)
Max_N_Servers : constant := 125 ;

   -- Maximum number of tasks that may be scheduled by the framework
Max_N_Tasks : constant := 25 ;

type Utilization_Set_Array is
  array (Positive_U_Index) of Utilization_Value;

type Utilization_Set is record
   Size  : U_Index := U_Index'First; --  '
   Value : Utilization_Set_Array;
end record;

type Critical_Sections_Array is
  array (Positive_CS_Index) of Critical_Section_Data;

type Critical_Sections is record
   Size  : CS_Index := CS_Index'First; --  '
   Value : Critical_Sections_Array;
end record;

type Shared_Obj_Handle is range 1 .. UINT32_MAX;

Null_Utilization_Set : constant Utilization_Set
  := (Size => 0,
      Value => (others => (C => Ada.Real_Time.Time_Span_Zero,
                           T => Ada.Real_Time.Time_Span_Zero)));

Null_Critical_Sections : constant Critical_Sections
  := (Size => 0,
      Value => (others => (Wcet => Ada.Real_Time.Time_Span_Zero,
                           Obj_Handle => Shared_Obj_Handle'First))); --  '

type Contract_Parameters is record
   Budget_Min : Ada.Real_Time.Time_Span := Ada.Real_Time.Time_Span_Zero;
   Period_Max : Ada.Real_Time.Time_Span := Ada.Real_Time.Time_Span_Zero;
   Budget_Max : Ada.Real_Time.Time_Span := Ada.Real_Time.Time_Span_Zero;
   Period_Min : Ada.Real_Time.Time_Span := Ada.Real_Time.Time_Span_Zero;
   The_Workload : Workload := Indeterminate;
```

```
   D_Equals_T              : Boolean := False;
   Deadline                : Ada.Real_Time.Time_Span := Null_Deadline;
   Budget_Overrun_Signal : POSIX_Signals.Signal := Null_Signal;
   Budget_Overrun_Signal_Data : POSIX_Signals.Signal_Data
     := Null_Signal_Data;
   Deadline_Miss_Signal  : POSIX_Signals.Signal := Null_Signal;
   Deadline_Miss_Signal_Data : POSIX_Signals.Signal_Data
     := Null_Signal_Data;

   The_Granularity : Granularity := Continuous;
   The_Utilization_Set : Utilization_Set := Null_Utilization_Set;
   The_Quality     : Quality := Quality'First; -- '
   The_Importance : Importance := Importance'First; -- '

   The_Preemption_Level : Preemption_Level
     := Preemption_Level'First; -- '
   The_Critical_Sections : Critical_Sections := Null_Critical_Sections;

   Policy              : Sched_Policy := NONE;
   The_Network_Id   : Network_Id := Null_Network_Id;
   Granted_Capacity_Flag : Boolean := False;
end record;

-- type Server_Id is new Natural range 0..Max_N_Servers;
type Server_Id is range 0 .. UINT32_MAX;

-- Maximum number of synchronization objects that can be created and
-- exist simultaneously to be used by the application
Max_N_Synch_Objects : constant := 5 ;

-- type SO_Index is range 0..Max_N_Synch_Objects;
-- subtype Positive_SO_Index is SO_Index range 1..SO_Index'Last; -- '

-- type Synch_Object_Handle is new Positive_SO_Index;
type Synch_Obj_Handle is range 0 .. UINT32_MAX;

function To_Signal_Data is new
  Ada.Unchecked_Conversion (Integer, POSIX_Signals.Signal_Data);

Null_Signal_Data : constant POSIX_Signals.Signal_Data
  := To_Signal_Data (0);

--
-- Other implementation dependent parameters
--

-- The Ada implementation in MaRTE OS will use the Application-Defined
-- Scheduling Interface (proposed to the POSIX standardization
-- committee), to create a fixed-priority-based scheduler that
```

```
-- operates under the rules of the FIRST scheduling framework.

-- In this implementation there are two special tasks:
--   - The application scheduler task, that implements the scheduler
--   - The service task, that is in charge of negotiating and
--     renegotiating contracts concurrently with the application

-- The following symbols are necessary to adapt the application to the
-- underlying fixed priority scheduler


-- Priority assigned to the application scheduler; it should be above
-- the priorities of the application tasks and of the service task,
-- and it should be at least 1 level below the maximum of the system
Scheduler_Priority : constant System.Priority := 30 ;

-- Real-time signal number reserved for the application scheduler
-- to manage its timers.
Scheduler_Signal : POSIX_Signals.Signal;
-- it is not a constant now because
-- Scheduler_Signal : constant POSIX_Signals.Signal
-- it has to be assigned in the body
-- := POSIX_Signals.Signal(SIGRTMIN );
-- The problem is that S I G R T M I N
--  is not a static value but a function

-- The highest priority that can be assigned to an application task,
-- it should be defined as one level less than the scheduler_priority
Highest_Thread_Priority : constant System.Priority
  := 30 -1 ; -- 30 -1 ; -- Scheduler_Priority-1;

-- The lowest priority that can be assigned to an application task,
-- it should be at least 1 level above the minimum of the system
Lowest_Thread_Priority :
  constant System.Priority := 3 ;
-- Each call to the functions that negotiate or renegotiate a contract
-- or that change the quality and importance generates a request for
-- the service task that we call a service job. This job will be
-- pending in a queue until executed by the service task. The
-- following symbol represents the maximum number of requests that can
-- be simultaneously queued.
Max_N_Service_Jobs :
  constant := 125 * 2 ; -- Max_N_Servers * 2;

-- In order to bound the background activity of the scheduler (i.e.,
-- the admission tests necessary for the negotiation and
-- re-negotiation of contracts), a service task has been defined. It
-- runs at a given priority level and has a budget and period
-- assigned.
```

```
-- Period of the service task (timespec)
Service_Task_Period : constant Ada.Real_Time.Time_Span
  := Ada.Real_Time."+"
  (Ada.Real_Time.To_Time_Span (Duration (0 )),
   Ada.Real_Time.Nanoseconds (100000000 ));

-- Budget of the service task (timespec)
Service_Task_Budget : constant Ada.Real_Time.Time_Span
  := Ada.Real_Time."+"
  (Ada.Real_Time.To_Time_Span (Duration (0 )),
   Ada.Real_Time.Nanoseconds (10000000 ));

-- Priority of the service task, it has to be lower than the
-- SCHEDULER_PRIORITY, and is set according to its period and the
-- expected response times for reconfiguration or tunning of the system.
Service_Task_Priority :
  constant System.Priority := 3 +1 ;

-- Maximum number of servers that can be simultaneusly waiting for being
-- signaled in a synchronization object
Max_N_Servers_In_Synch_Object :
  constant := 4 ;

-- Maximum number of events that can be pending to be
-- signaled in a synchronization object
Max_N_Events_In_Synch_Object :
  constant := 100 ; -- 254

-- Maximum number of pending replenishments in each sporadic server
Max_N_Pending_Replenishments : constant := 250 ;

-- This function must be supplied by the user to map the preemption
-- level values given in the contracts for the servers, to priority
-- values in the range that is allowed by the present implementation
-- for application tasks. The value returned by the function must
-- fit in the interval defined by the constants:
-- [LOWEST_TASK_PRIORITY, HIGHEST_TASK_PRIORITY]

function Priority_Map
  (Plevel : in Preemption_Level)
  return System.Priority;


type Contracts_Array is
  array (Positive_C_Index) of Contract_Parameters;

type Contract_Set is record
   Size  : C_Index := C_Index'First; -- '
   Value : Contracts_Array;
end record;
```

```
Null_Contract_Parameters : Contract_Parameters;

Null_Contract_Set : constant Contract_Set
  := (Size => 0, Value => (others => Null_Contract_Parameters));

type Servers_Array is
  array (Positive_S_Index) of Server_Id;

type Server_set is record
   Size  : S_Index := S_Index'First; -- '
   Value : Servers_Array;
end record;


Null_Server_set : constant Server_set
  := (Size => 0, Value => (others => Server_Id'First)); -- '

type Target_Windows_Array is
  array (Positive_TW_Index) of Target_Window_Data;

type table_driven_params is record
   Size  : TW_Index := TW_Index'First; -- '
   Value : Target_Windows_Array;
end record;

Null_table_driven_params : constant table_driven_params
  := (Size => 0,
      Value => (others => (The_Start => Ada.Real_Time.Time_Span_Zero,
                           the_end => Ada.Real_Time.Time_Span_Zero,
                           Comp_Time => Ada.Real_Time.Time_Span_Zero)));

subtype Hiearchical_Module_Supported is Boolean range False .. False;
subtype Distributed_Module_Supported is Boolean range True .. True;
subtype Distributed_Spare_Capacity_Module_Supported
is Boolean range False .. False;
subtype Spare_Capacity_Module_Supported is Boolean range True .. True;
subtype Implementation_Specific_Module_Supported
is Boolean range True .. True;
subtype Dynamic_Reclaiming_Module_Supported is Boolean range False .. False;
subtype Shared_Objects_Module_Supported is Boolean range True .. True;


-- Budget of the round robin slice (timespec) for the background
RR_Slice_CPU_Time : constant Ada.Real_Time.Time_Span
  := Ada.Real_Time."+"
  (Ada.Real_Time.To_Time_Span (Duration (0 )),
   Ada.Real_Time.Nanoseconds (100000000 ));
```

C.2 FSF.Distributed private part

```
private

   type unsigned_32 is range 0 .. Fsf.UINT32_MAX;

   type Implementation_Defined_Node_Address (Net : Network_id) is record
      case net is
         -- when 1 => rt_ep : unsigned_32; -- Implementation dependent
         -- when others => null;
         when others => default : unsigned_32;
      end case;
   end record;

   type Implementation_Defined_Port_Number (Net : Network_id) is record
      case net is
         -- when 1 => rt_ep : Natural; -- Implementation dependent
         -- when others => null;
         when others => default : Natural;
      end case;
   end record;

   -- This operation is used by the implementation to convert the node
   -- identification (used in the creation of a send endpoint) to the
   -- specified network communication provider dependent address
   function
     get_node_address
     (the_network_id : in Fsf.Network_id := Fsf.Default_Network_id;
      node           : in Node_address)
     return implementation_defined_node_address;

   -- This operation is used by the implementation to get the local node
   -- address as it is to be used by the communication provider in the
   -- network specified
   function
     get_local_node_address
     (the_network_id : in Fsf.Network_id := Fsf.Default_Network_id)
     return Implementation_defined_node_address;

   -- This operation is used by the implementation to convert the port
   -- number given in the creation of a receive endpoint to the
   -- specified network communication provider dependent port
   function
     get_port
     (the_network_id : in Fsf.Network_id := Fsf.Default_Network_id;
      the_port       : in Port)
     return Implementation_defined_port_number;

   -- This two types are indexes to internally created arrays with
   -- the necessary information
```

```
type Send_endpoint is new Natural;

type Receive_endpoint is new Natural;
```