SPARK Ada programming for ARM-based device (BeagleBoard-xM)

Jakub Jedryszek
Kansas State University
Manhattan, Kansas
jakub.jedryszek@gmail.com

Ashish Pinninti
Kansas State University
Manhattan, Kansas
ashishpinninti@gmail.com

This paper presents running single and multithreaded SPARK Ada programs on BeagleBoard-xM device. It describes basics of SPARK Ada programming language. A language dedicated for safety-critical systems. There is an analysis of its concurrency (tasking) features supported by RavenSPARK profile. There are two examples of single- and multitasking applications presented: Odometer and Simple PCA Pump (along with its Uppaal models).

1 Introduction

SAnToS Lab conducts research on Medical Devices (type of safety-critical systems) and software verification. Currently, there is an effort to create prototype device: PCA Pump (Patient Controlled Analgesia). Thus a platform for prototyping PCA Pump and future prototypes is needed. Usually, every embedded device comes with its own, customized compiler. SAnToS Lab created Sireum distribution - software analysis platform for SPARK Ada, which takes advantage of GNAT compiler (front-end for gcc compiler [9]). Therefore, GNAT compiler (more precisely: GNAT cross-compiler for ARM-based devices) is used here for device prototyping. The main objective of this project is to investigate SPARK Ada as a language for ARM-based device: BeagleBoard-xM, especially: its concurrency capabilities.

BeagleBoard-xM is only temporary platform. Electrical and Computer Engineering department at Kansas State University is working on hardware realization of devices suitable for medicine. Ultimately, medical devices software created by SAnToS Lab will be run on dedicated devices created by ECE department.

1.1 SPARK Ada

SPARK programming language is based on Ada. Thus, it is usually referred as SPARK Ada. SPARK is a programming language and static verification technology designed specifically for the development of high integrity software [8]. First version was designed over 20 years ago. SPARK excludes some Ada constructs (i.e. pointers, dynamic memory allocation or recursion) to make static analysis feasible. Additionally SPARK contains tool-set for Software Verification:

- Examiner analyzes code and ensures that it conforms to the SPARK language; also verifies program to some extent using Verification Conditions (e.g. array index out of range, type range violation, division by zero, numerical overflow etc.)
- Simplifier simplifies Verification Conditions generated by Examiner
- Proof Checker assures the Verification Conditions

Submitted to:
CIS 721 - Real-Time Operating Systems Project

Using SPARK, a developer takes a Z specification and builds a step-by-step refinement of SPARK code. For each refinement step a tool is used to produce verification conditions (VC's), which are mathematical theorems. If the VC's can be proved then the refinement step will be valid,otherwise, the refinement step can be erroneous. Sample Verification Condition contains checks for:

- array index out of range
- type range violation
- division by zero
- numerical overflow

First version of SPARK (SPARK 83) was based on Ada 83. The second version (SPARK 95) on Ada 95. Current version SPARK 2005 is based on Ada 2005. It is a subset of Ada 2005 with annotations. The annotation language supports flow analysis and formal verification. Annotations are encoded in Ada comments (using the prefix –#). It makes every SPARK 2005 program, a valid Ada 2005 program.

SPARK 2014 (based on Ada 2012) is under development. There is partial tool support (in GNAT Programming Studio), but some language features are still not supported. It is worth to mention, that Ada 2012 contains code contracts. Thus SPARK 2014 is just a subset of Ada 2012. There are no additional constructs such as annotations in SPARK 2005, because now, contracts are part of the language.

In real-world applications, the embedded critical components are written in SPARK while the non-critical components are written in Ada.

The crucial part of SPARK Ada are code contracts specified as annotations. Listing 1 presents simple procedure with code contracts. It increments variable given as parameter by 1. The `derives` specified variable dependency. It future value depends on its current value. There is pre condition saying that the value has to be lower than maximum value of Integer type. There is also post condition, which states that the value of variable (given as parameter) after the procedure execution has to be equal to its previous value incremented by 1.

```
procedure Increment (X : in out Integer);
--# derives X from X;
--# pre X < Integer'Last;
--# post X = X~ + 1;
```

Listing 1: Sample SPARK procedure with code contracts

1.2 RavenSPARK

RavenSPARK is subset of the SPARK Ravenscar Profile (which is subset of Ada tasking). The Ravenscar Profile provides a subset of the tasking facilities of Ada95 and Ada 2005 suitable for the construction of high-integrity concurrent programs [12].

The Ravenscar Profile is a subset of the tasking model, restricted to meet the real-time community requirements for determinism, schedulability analysis and memory-boundedness, as well as being suitable for mapping to a small and efficient run-time system that supports task synchronization and communication, and which could be certifiable to the highest integrity levels. The concurrency model promoted by the Ravenscar Profile is consistent with the use of tools that allow the static properties of programs to be verified. Potential verification techniques include information flow analysis, schedulability analysis, execution-order analysis and model checking. These techniques allow analysis of a system to be performed throughout its development life cycle, thus avoiding the common problem of finding only during system integration and testing that the design fails to meet its non-functional requirements. [3]

1.3 BeagleBoard-xM

BeagleBoard-xM is an open-source hardware single-board computer produced by Texas Instruments. It has a configuration of 1GHz ARM Cortex-A8, 512 MB RAM, 4 USB ports, and SD connector, which makes it capable of running Linux Operating System. These properties, along with General Purpose Input/Output ports (4 PWM enable), make BeagleBoard-xM a very potential board for creating prototypes of devices.

Pulse Width Modulation (PWM) enables control of analog circuits with digital outputs. To achieve higher speed of actuator, we need to give higher frequency and/or higher duty cycle. Duty cycle is percentage of period in which a signal is active.

2 Literature review

There is very few resources for SPARK Ada developers accessible online. It is very hard to find some examples of working code. The real-life examples are almost impossible to find, because they are keep secret by companies. The best resources for SPARK Ada development is Barnes' book [4], SPARK user manual [1] and SPARK user guide [2]. While, former are more language-specific resources, "Industrial Experience with SPARK" [6] and "Rail, SpaceSecurity: Three Case Studies for SPARK 2014" [7] are more practical. Good introduction to SPARK is "An IntegratedApproach to High Integrity Software Verification" paper [10].

For RavenSPARK, we find useful "Guide for the use of the Ada Ravenscar Profile in high integrity systems" [3] and The SPARK Ravenscar Profile documentation [12]. During development, SPARK Examiner was also useful.

The BeagleBoard ports interaction is well described in reference manual [5]. However, various online resources were necessary to implement it.

3 Running SPARK Ada programs on BeagleBoard-xM

3.1 Hello, World

To run SPARK Ada program on BeagleBoard-xM, first we need to cross-compile it. As an IDE for SPARK Ada development, we use GNAT Programming Studio[1]. To create "Hello, World!" application, we created new Ada project (choosing Project/New... from the menu). Then we created main.adb file with procedure Main printing "Hello, World!" in standard output. The code is on listing 2. It is valid Ada 2005 and Ada 2012 code.

```
with Ada.Text_IO;

procedure Main
is
begin
    Ada.Text_IO.Put_Line("Hello, World!");
end Main;
```

Listing 2: "Hello World" in Ada

[1] http://libre.adacore.com/tools/gps/

To run it, we need to add created file to list of 'main files' in Project/Edit Project Properties (figure 3.1), tab: Main files (figure 3.1). Then, we can 'Build All' and 'Run Main' clicking appropriate toolbar buttons. The main file has to be always specified in order to compile and link application, which will be runnable.

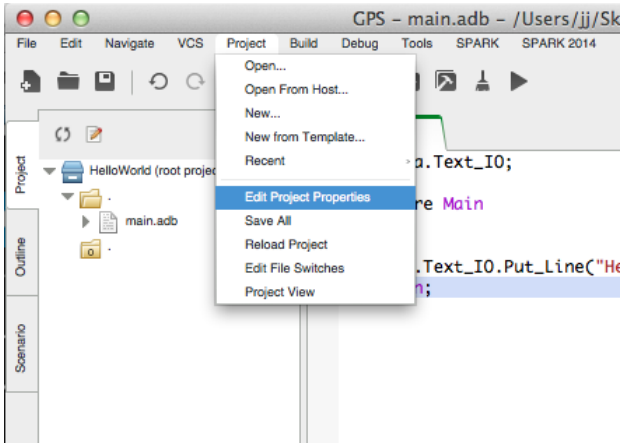

Figure 1: Edit Project Properties

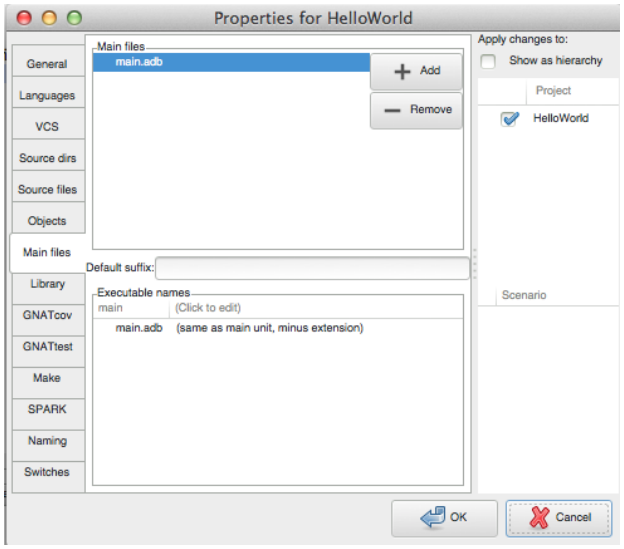

Figure 2: Project Main files

To run it on BeagleBoard-xM we have to use cross-compiler and issue following command: `arm -linux-gnueabi-gnatmake -d -Phelloworld.gpr` (helloworld.gpr is GNAT Programming Studio project file). We can specify additional flags in command line or directly in project file (manually or through GNAT Programming Studio Interface).

3.2 Odometer

Odometer is a simple SPARK Ada program, which implements basic functions of standard Odometer. Listing 3 shows an interface of Odometer in SPARK 2005.

```
package Odometer
--# own
--# Trip,        -- number of meters so far on this trip (can be reset to 0).
--# Total        -- total meters traveled of vehicle since the last factory-reset.
--#   : Natural; -- has range 0 .. Integer'Last.
--# initializes Trip,
--#             Total;
is

    procedure Zero_Trip; -- sets Trip to 0 and clears all saved Trip marks.
    --# global out Trip;
    --# derives Trip from ;
    --# post Trip = 0;

    function Read_Trip return Natural; -- returns value of Trip.
    --# global in Trip;
    --# return Trip;

    function Read_Total return Natural; -- returns value of Total
    --# global in Total;
    --# return Total;

    procedure Inc; -- increments each of Trip and Total by 1.
    --# global in out Trip, Total;
    --# derives Trip from Trip & Total from Total;
    --# pre Trip < Integer'Last and Total < Integer'Last;
    --# post Trip = Trip~ + 1 and Total = Total~ + 1;

end Odometer;
```

Listing 3: SPARK 2005 code: Odometer

There are 4 subprograms (2 procedures and 2 functions), which are globally available (through other packages and program units):

- Zero_Trip procedure reset Odometer to 0
- Read_Trip function returns current distance
- Read_Total function returns total distance traveled
- Inc procedure - increment total and current distance by 1

Annotation `global` means that subprogram uses some global variable. Postfix in, out or in out means that particular variable is read, write or read and write respectively. Annotation `derives` says that some variable value depends on other variables. E.g. in procedure `Inc` variable `Trip` is dependent on its current value (before procedure call). Annotations `pre` and `post` define pre- and postconditions of procedure. We can see, that in `Zero_Trip` procedure postcondition requires that variable `Trip` is equal to 0. In procedure `Inc`, postconditions requires that variables `Trip` and `Total` are incremented by 1 (is the value of variable before procedure call). Annotation `own` expose private variables for use in public methods specification. Annotation `initializes` ensures that given variables are initializes. It will allows SPARK tools to perform static analysis.

We can check correctness of Odometer running SPARKMake (3.2), then setting generated spark.idx file in Properties/Switches/Examiner as SPARK index file (3.2).

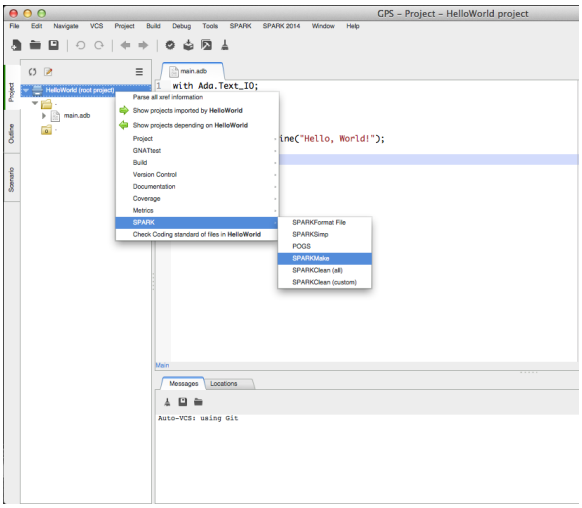

Figure 3: Run SPARK Make

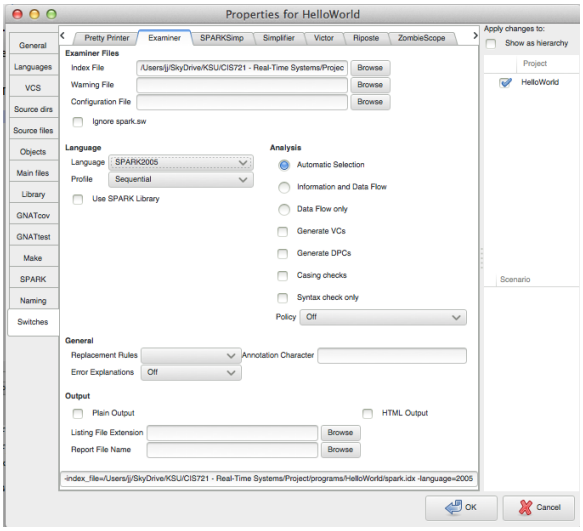

Figure 4: Examiner Properties

Odometer in SPARK 2005 works fine on BeagleBoard-xM. In order to test SPARK 2014 program, we translated Odometer to SPARK 2014 syntax. Listing 4 is package spacification of Odometer in SPARK 2014.

```
package Odometer
with Abstract_State => (Trip_State, Total_State)
is

   function Trip_State return Integer
```

```
      with Convention => Ghost,
      Global => (Input => Trip_State);

   function Total_State return Integer
      with Convention => Ghost,
      Global => (Input => Total_State);

   procedure Zero_Trip -- sets Trip to 0
      with Global => (Output => (Trip_State)),
      Depends => (Trip_State => null),
      Post => Trip_State = 0;

   function Read_Trip return Natural -- returns value of Trip.
      with Global => (Input => Trip_State),
      Post => Read_Trip'Result = Trip_State;

   function Read_Total return Natural -- returns value of Total
      with Global => (Input => Total_State),
      Post => Read_Total'Result = Total_State;

   procedure Inc -- increments each of Trip and Total by 1.
      with Global => (In_Out => (Trip_State, Total_State)),
      Depends => (Trip_State => Trip_State, Total_State => Total_State),
      Pre => Trip_State < Integer'Last and Total_State < Integer'Last,
      Post => Trip_State = Trip_State'Old + 1 and Total_State = Total_State'Old + 1;

end Odometer;
```

Listing 4: SPARK 2014 code: Odometer

4 SPARK Ada Tasking on BeagleBoard-xM

The main goal of the research is to study the concurrency in SPARK Ada regarding target platform (BeagleBoard-xM) and existing GNAT cross-compiler. In Ada World, concurrency is referred as tasking and the task is the same construct as the thread in other programming languages.

In SPARK 2005 multitasking is possible with Ravenscar Profile. Default profile - sequential - does not enable tasking. In other words, SPARK tools cannot analyze and reason about programs if Ravenscar profile flag is not provided.

In SPARK 2014 - for now tasking is not possible. It's part of SPARK 2014 roadmap to include support for tasking in the future, although likely not this year.

4.1 Ada multitasking application

Listing 5 is a simple multitasking application in Ada 2005. It is also valid code for Ada 2012. There are 3 tasks:

- Main task
- S (type: Seconds) - simple counter printing numbers form 1 to 10 in every second
- T (type: Tenth_Seconds) - simple counter printing numbers from 0.1 to 10 in every 0.1 second

```
with Ada.Text_IO;
use Ada.Text_IO;
with Ada.Float_Text_IO;

procedure Main is

    task type Seconds is
    end Seconds;

    task type Tenth_Seconds is
    end Tenth_Seconds;

    S : Seconds;
    T : Tenth_Seconds;

    task body Seconds is
    begin
        for I in 1..10 loop
            delay Standard.Duration(1);
            Put_Line(Integer'Image(I));
        end loop;
    end Seconds;

    task body Tenth_Seconds is
    begin
        for I in 1..100 loop
            delay 0.1;
            Ada.Float_Text_IO.Put(Float(I)/Float(10), AFT=>2, EXP=>0);
            Put_Line("");
        end loop;
    end Tenth_Seconds;

begin
    Put_Line("Started");
end Main;
```

Listing 5: Simple multitask application in Ada

4.2 SPARK Ada multitasking application

Multitasking SPARK Ada application can be created only using RavenSPARK. Multitasking Odometer, is extended version of Odometer from listing 3. It has additional variable (Speed), procedure (Set_Speed) and new task: Drive. Thus, in total it has two tasks:

- Main
- Drive - increase Total and Trip by Speed (m/s) in every second

The interface of extended odometer is shown on listing 6.
There are two ways to access protected variable in task body:

- It has to be protected object
- It has to be atomic type

```
--# inherit Ada.Real_Time;

package Odometer
--# own Trip : Distance;
--#     Total : Distance;
--#     Speed : Meters_Per_Second;
--#     task d : Drive;
--# initializes Trip,
--#             Total,
--#             Speed;
is
    type Distance is range Natural'First .. Natural'Last;
    pragma Atomic (Distance);

    type Meters_Per_Second is range Natural'First .. Natural'Last;
    pragma Atomic(Meters_Per_Second);

    procedure Zero_Trip; -- sets Trip to 0 and clears all saved Trip marks.
    --# global out Trip;
    --# derives Trip from ;
    --# post Trip = 0;

    function Read_Trip return Distance; -- returns value of Trip.
    --# global in Trip;
    --# return Trip;

    function Read_Total return Distance; -- returns value of Total
    --# global in Total;
    --# return Total;

    procedure Inc; -- increments each of Trip and Total by 1.
    --# global in out Trip, Total;
    --# derives Trip from Trip & Total from Total;
    --# pre Trip < Distance'Last and Total < Distance'Last;
    --# post Trip = Trip~ + 1 and Total = Total~ + 1;

    procedure Set_Speed(New_Speed : Meters_Per_Second);
    --# global out Speed;
    --# derives Speed from New_Speed;
    --# post Speed = New_Speed;
private

    task type Drive
    --# global in     Speed;
    --#        in out Trip;
    --#        in out Total;
    --#        in     Ada.Real_Time.ClockTime;
    is
        pragma Priority(10);
    end Drive;

end Odometer;
```

Listing 6: Multitasking Odometer

Protected variables may not be used in proof contexts. Thus, if we try to use protected variable in proofs (pre- or postcondition), then we get Semantic Error 940 - Trip is a protected own variable. To preserve pre- and postconditions from original Odometer, atomic types (Distance and Meters_Per_Second) has been used.

5 Simple PCA Pump

Patient Controlled Analgesia (PCA) pump is a medical device, which allows the patient to self-administer small doses of narcotics (usually Morphine, Dilaudid, Demerol, or Fentanyl). PCA pumps are commonly used after surgery to provide a more effective method of pain control than periodic injections of narcotics. A continuous infusion (called a basal rate) permits the patient to receive a continuous infusion of pain medication. Patient can also request additional boluses, but only in specified intervals. It prevents from overinfusion. In addition to basal and patient bolus, clinician can also request bolus called clinician bolus or square bolus. All information needed for Pump to operate should be specified by physician in the prescription, which is entered into PCA Pump. The prescription contains:

- basal rate
- volume to be infused during patient bolus
- minimum time between patient boluses
- maximum drug amount allowed per hour

There can be situations in which the maximum drug amount induced may exceed the limit. E.g. when clinician issues too long square bolus. In such case, the Pump is switched to Keep Vein Open (KVO) mode, which has 1ml/hr drug rate. From this state pump has to be restarted by clinician. In Summary, Simple PCA Pump has following modes:

- Stopped
- Basal rate
- Bolus
- Clinician bolus (Square bolus)
- Keep Vein Open (KVO)

The pump has three tasks in total:

- main task (requesting boluses, displaying volume infused etc.)
- rate controller - to control the speed of infusion rate
- max drug per hour watcher - to control overinfusion

The main PCA Pump Package can be verified using SPARK tools. The Pump engine (motor) is separated entity. It has an interface allowing request infusion of 0.1 ml of drug.

Simple PCA Pump was developed based on Requirements Documentation created by Brian Larson, John Hatcliff and Partice Chalin [11].

5.1 UPPAAL Verification

We can reinforce SPARK's verification capability by modeling the PCA Pump in UPPAAL.

Depending on the functionalities of PCA pump and its environment, we have designed the system with four templates: Pump, Patient, Clinician, Max Drug Per Hour Watcher. We have different states in which the Pump can stay namely: Basal, Bolus, Stopped, Square Bolus, KVO etc. Similarly Patient can be present in Bolus, No Bolus, and KVO states, while Clinician can be present in No Square Bolus, Square Bolus, and KVO states. The watcher will help in monitoring the drug delivery. It enters KVO state once the drug delivered exceed the limit. We have used broadcasting channels such as start, bolus, end bolus, square bolus, end square bolus, go to kvo, restart to synchronize more than two templates.

We have tested the model for few queries such as `E<> Watcher.KVO`, `A<> PCAPump.Bolus`, `E<> PCAPump.Bolus`, `E<> PCAPump.Square`, `E<> PCAPump.Waiting`, `E<> PCAPump.Interrupted` etc. The results of each query made sense. For example `E<> Watcher.KVO` query went green meaning that there exists some path eventually for the Watcher to enter KVO state.

6 Summary

To run SPARK Ada programs on BeagleBoard-xM we took advantage of GNAT cross-compiler. It compiles and links SPARK Ada 2005 single and multitasking programs without any problems. In case of SPARK Ada 2014 only single tasking programs are allowed. Tasking features are not yet in SPARK 2014, which is under development. After successful compilation and linking, we did not have any issues with running programs on BeagleBoard-xM. Moreover, their behaviour on Intel processor (PC or MacBook) is the same like on ARM device.

References

[1] AdaCore & Altran UK Ltd (2011-2014): *SPARK 2014 Reference Manual*. Available at `http://docs.adacore.com/spark2014-docs/html/lrm`.

[2] AdaCore & Altran UK Ltd (2011-2014): *SPARK 2014 Toolset Users Guide*. Available at `http://docs.adacore.com/spark2014-docs/html/ug`.

[3] Tullio Vardanega Alan Burns, Brian Dobbing (2004): *Guide for the use of the Ada Ravenscar Profile in high integrity systems*. *ACM SIGAda Ada Letters* 24(2), pp. 1–74.

[4] John Barnes (2013): *SPARK - The Proven Approach to High Integrity Software*. Altran.

[5] BeagleBoard.org (2010): *BeagleBoard-xM Rev C System Reference Manual*. Available at `http://beagleboard.org/static/BBxMSRM_latest.pdf`.

[6] Roderick Chapman (2000): *Industrial Experience with SPARK*. *ACM SIGAda Ada Letters - special issue on presentations from SIGAda 2000* XX(4), pp. 64–68.

[7] Claire Dross, Pavlos Efstathopoulos, David Lesens, David Mentre & Yannick Moy (2014): *Rail, Space Security: Three Case Studies for SPARK 2014*. In: *ERTS 2014: Embedded Real Time Software and Systems*.

[8] Michael B. Feldman (2012): *Who's Using Ada? Real-World Projects Powered by the Ada Programming Language*. Available at `http://www.seas.gwu.edu/~mfeldman/ada-project-summary.html`.

[9] Bartomiej Horn (2009): *Ada'05 compiler for ARM based systems*. thesis, Technical University of Lodz, Poland.

[10] Andrew Ireland, Bill J. Ellis, Andrew Cook, Roderick Chapman & Janet Barnes (2006): *An Integrated Approach to High Integrity Software Verification*. *Journal of Automated Reasoning* 36(4), pp. 379–410.

[11] Brian R. Larson, John Hatcliff & Patrice Chalin (2013): *Open Source Patient-Controlled Analgesic Pump Requirements Documentation*. In: *Software Engineering in Health Care (SEHC), 2013 5th International Workshop*, Institute of Electrical and Electronics Engineers (IEEE), pp. 28–34.

[12] SPARK Team (2012): *The SPARK Ravenscar Profile*. Available at `http://docs.adacore.com/sparkdocs-docs/Examiner_Ravenscar.htm`.