

# **An Ada Interface to Lego Mindstorms**

Barry Fagin

Department of Computer Science

US Air Force Academy

Colorado Springs, CO

719-333-3340

[barry.fagin@usafa.af.mil](mailto:barry.fagin@usafa.af.mil)

## 1.0 Introduction

This article presents an Ada interface to the Lego Mindstorms™ RCX “brick”, the core element of a commercial product that emerged from a collaboration between the Lego Corporation and MIT’s Media Laboratory. Since its arrival on the market in 1998, it has considerable interest in the computing community: a variety of operating systems and language interfaces are available for it, all provided free of charge by people who simply wanted to find out how the RCX worked.

The interface described here is in use at the Air Force Academy as part of an experiment in computer science education. Students with no prior programming background are given a pre-built Mindstorms robot and a series of programming challenges, which they then attempt to implement using Ada. Our hope is that the experience of programming robots will provide an effective, efficient, and enjoyable method for conveying essential computer science concepts. We are currently attempting to assess its effectiveness, and hope to present our results at a future date.

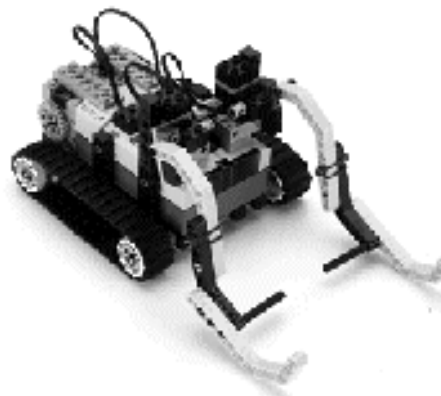
## 2.0 The Lego RCX Platform

The RCX brick is shown below:



**Figure 1: The Lego RCX**

It contains a Hitachi microcontroller, three input ports numbered 1, 2, and 3, and three output ports labeled A, B, and C. Inputs can be connected to a variety of sources, including touch sensors (that detect whether or not a button has been pressed), light sensors (that measure the intensity of reflected light) and even temperature sensors. Outputs are usually connected to motors, to provide mobility, a grabber arm, or a similar moving component. The sensors, motors, and other Lego components are included with the Mindstorms kit, which is used to build a device around the brick itself. One example is shown below:



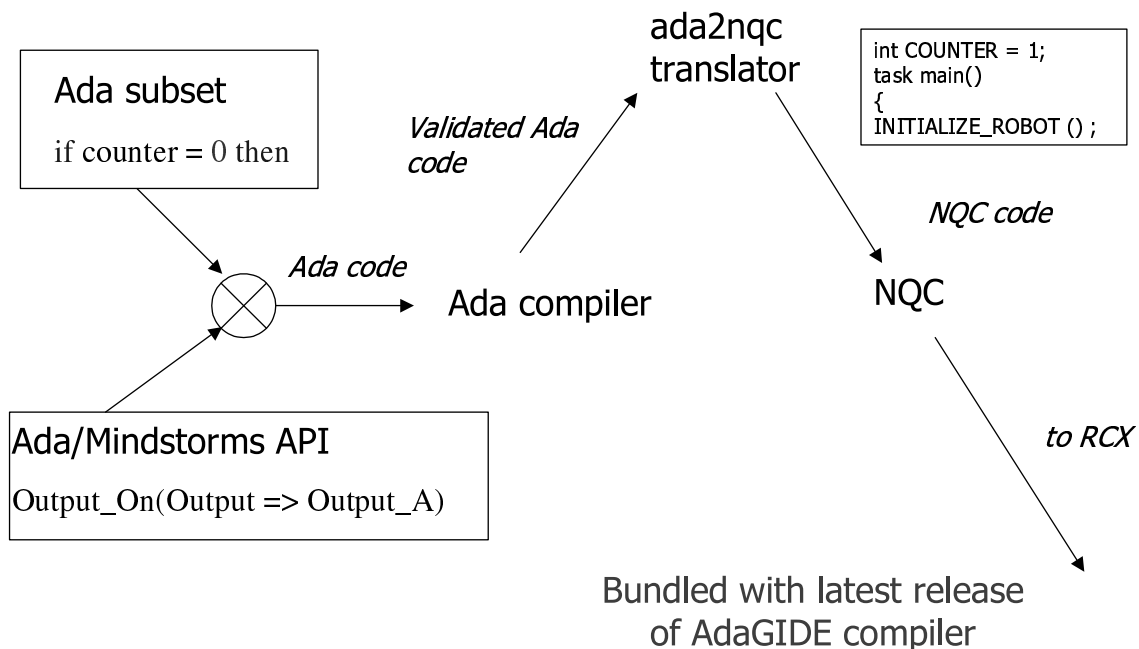
**Figure 2: A Mindstorms Robot**

### 3.0 The Ada/Mindstorms Programming Environment

Ada/Mindstorms 1.0 consists of two files, lego.ads and lego.adb, and a translator known as ada2nqc. It also requires access to an Ada compiler, which must be run on the source program before the translator is run. The translator performs some semantic checks that a regular compiler cannot catch (many features of Ada, for example, cannot be supported in the RCX hardware). It then converts the Ada program to a language known as NQC, for “Not Quite C”.

NQC was developed by Dave Baum, and is a functioning high-level language execution environment for Lego Mindstorms [Ba99]. While NQC is well suited for experienced programmers and curious hackers, it is not an attractive language for teaching programming, particularly for students with no programming background. By providing an Ada interface, we are able to use a more controlled, rigorous, and forgiving programming environment, one we believe to be better suited to computer science education.

The complete Ada/Mindstorms programming process is shown below:



**Figure 3: Ada/Mindstorms 1.0**

Note that Ada/Mindstorms 1.0 is bundled into the latest release of AdaGide, which permits the building of executables for multiple targets (available from [http://www.usafa.af.mil/dfcs/bios/mcc\\_html/adagide.html](http://www.usafa.af.mil/dfcs/bios/mcc_html/adagide.html)). The base code for the ada2nqc translator was produced using the Ada Generator of Object-Oriented Parsers (AdaGOOP), developed by Carlisle [Ca00].

## 4.0 The Ada Subset Supported by Ada/Mindstorms

Many of the more elaborate features of Ada cannot be executed on the RCX hardware. Others would add substantial complexity to the translation software, and are planned for later releases. Thus Ada/Mindstorms 1.0 programs are currently subject to the following restrictions:

- 1) All identifiers and literals must be integers
- 2) Procedures may only appear inside the main procedure
- 3) No separate compilation
- 4) No standard packages
- 5) No user-defined types

More detailed restrictions are described in the Ada/Mindstorms 1.0 User's Manual, available at <http://www.usafa.af.mil/dfcs/adamindstorms.htm>. Some of the material in this paper is also available there.

## 5.0 A Sample Program

Below is a simple Ada/Mindstorms program, written for a robot with two motors connected to outputs A and C and a bumper with a touch sensor connected to input A. This program will make the robot go forward until it bumps into something, then go back for three seconds, turn, and go forward again. The direction of the turn alternates each time the bumper is pressed.

```
with Lego;
use Lego;

procedure Test is
  Left : constant Integer := 0;
  Right : constant Integer := 1;

  procedure Go_Forward is
  begin
    Output_On_Forward(Output => Output_A);
    Output_On_Forward(Output => Output_C);
  end Go_Forward;

  procedure Go_Back (Tenths_Of_A_Second : in Integer ) is
  begin
    Output_On_Reverse(Output => Output_A);
    Output_On_Reverse(Output => Output_C);
    Wait (Hundredths_Of_A_Second => Tenths_Of_A_Second * 10);
    Output_Off(Output => Output_A);
    Output_Off(Output => Output_C);
  end Go_Back;

  procedure Turn (Direction : in Integer ) is
  begin
    if Direction = Left then
      Output_Off(Output => Output_A);
      Output_Power(Power => Power_Low, Output => Output_C);
      Output_On_Forward(Output => Output_C);
    end if;
  end Turn;
end Test;
```

```

        Wait (Hundredths_Of_A_Second => 100);
        Output_Off(Output => Output_C);
    else --right
        Output_Off(Output => Output_C);
        Output_Power(Output => Output_A,Power => Power_Low);
        Output_On_Forward(Output => Output_A);
        Wait (Hundredths_Of_A_Second => 100);
        Output_Off(Output => Output_A);
    end if;
end Turn;

procedure Initialize_Robot is
begin
    Config_Sensor(Sensor => Sensor_1, Config => Config_Touch);
    Output_Power(Output => Output_A,Power => Power_Low);
    Output_Power(Output => Output_C,Power => Power_Low);
end Initialize_Robot;

counter : integer := 1;
begin
    Initialize_Robot;
    Go_Forward;
    loop
        --touch sensor pressed?
        if Get_Sensor_Value(Sensor => Sensor_1) = 1 then
            Go_Back(Tenths_Of_A_Second => 30);
            if counter = 0 then
                Turn(Direction => Right);
                counter := 1;
            else
                Turn(Direction => Left);
                counter := 0;
            end if;
            Go_Forward;
        end if;
    end loop;
end Test;

```

**Figure 4: A Sample Ada/Mindstorms Program**

## 6.0 Where to Obtain Ada/Mindstorms 1.0, and Future Work

Ada/Mindstorms 1.0 is available for PC's running Windows; see <http://www.usafa.af.mil/dfcs/adamindstorms.htm> for details. Source code for the ada2nqc translator is available from the author.

We are currently experimenting with efforts to use Ada/Mindstorms 1.0 and RCX projects to teach computer science. Work in progress includes an investigation into which topics are best suited for robotics-based instruction, and an assessment of the effectiveness of robotics-based instruction for teaching computer science.

Future plans for Ada/Mindstorms include support for arrays, separate compilation, and tasks. We also plan to construct a simulator, by replacing the stubs for the API procedures in lego.adb with actual code.

## 7.0 Acknowledgements

This work was funded by a grant from the Institute for Information Technology Applications, whose support is gratefully acknowledged. Readers with questions about Ada/Mindstorms 1.0 are invited to visit the Ada/Mindstorms web site, review other work on this topic [Fa00], or to contact the author.

## 8.0 References

[Ba99] Baum, D., *The NQC web site* ([http:// www.enteract.com/~dbaum/nqc/](http://www.enteract.com/~dbaum/nqc/))

[Ca00] Carlisle, M., An Automatic Object-Oriented Parser Generator for Ada, *Ada Letters*, Vol 20 No 2, June 2000, pp 57-63.

[Fa00] Fagin, B., Using Ada-Based Robotics to Teach Computer Science, *Proceedings of the 5<sup>th</sup> Annual Conference on Innovation and Technology in Computer Science Education*, Helsinki, Finland, July 2000, pp 148-151.

[Kn99] Knudsen, J., *The Unofficial Guide to LEGO MINDSTORMS Robots*, O'Reilly & Associates, ISBN: 1565926927

## Appendix: lego.ads<sup>1</sup>

```
-----  
--  
-- File:           Lego.ads  
-- Description:   Package to implement LEGO commands  
--  
--  
--  
-----  
  
package Lego is  
  
    type Ada_Mindstorms_Procedure is (  
        Set_Sensor,  
        Set_Sensor_Mode,  
        Set_Sensor_Type,  
        Clear_Sensor,  
        Get_Sensor_Value,  
        Output_On,  
        Output_Off,  
        Output_Float,  
        Output_Forward,  
        Output_Reverse,  
        Output_Toggle,  
        Output_Power,  
    );  
  
end Lego;
```

---

<sup>1</sup> Lego.adb contains only empty procedure bodies for the code indicated here.

```

    Wait,
    Play_Sound,
    Play_Tone,
    Stop_All_Tasks,
    Send_Message,
    Clear_Message,
    Config_Sensor,
    Output_On_Forward,
    Output_On_Reverse,
    Output_On_For,
    Set_Output_Mode,
    Set_Output_Direction
);

-- type used to specify a sensor port (1..3) on the RCX.
type Sensor_Port is (Sensor_1, Sensor_2, Sensor_3);

-- type used to specify an RCX output port (A B or C).
type Output_Port is (Output_A, Output_B, Output_C);

-- type used to specify how the sensor input is handled.
type Configuration is
    (Config_Touch,
     Config_Light,
     Config_Pulse,
     Config_Rotation,
     Config_Celsius,
     Config_Fahrenheit,
     Config_Edge);

-- type used to specify an RCX output mode
type Output_Mode is
    (Output_Mode_On,
     Output_Mode_Off,
     Output_Mode_Float);

-- type used to specify an RCX output direction
type Output_Direction is
    (Output_Direction_Forward,
     Output_Direction_Reverse,
     Output_Direction_Toggle);

-- type used to specify the actual sensor type (hardware).
type Sensor_Type is
    (Type_Touch,
     Type_Temperature,
     Type_Light,
     Type_Rotation);

-- type used to specify the power level of an output.
type Power_Type is range 0..7;

-- type used to specify the type of sound desired.
type Sound is
    (Click,
     Double_Beep,
     Down,

```

```

        Up,
        Low_Beep,
        Fast_Up);
-- type used to specify a message number.
type Message is range 1 .. 255;
-- type used to specify a duration in 10ths of a second.
type Duration is new Positive;
-- type used to specify a frequency.
type Frequency is new Positive;
-- type used to hold the value of the sensor.
type Sensor_Value is new Integer;

--type for sensor modes
type Sensor_Mode is (
    Mode_Raw,
    Mode_Boot,
    Mode_Edge,
    Mode_Pulse,
    Mode_Percent,
    Mode_Celsius,
    Mode_Fahrenheit,
    Mode_Rotation);

--power level definitions, supplied as a convenience
Power_Low : constant Power_Type := 1;
Power_Half : constant Power_Type := 4;
Power_High : constant Power_Type := 7;

-----
-- Procedures      --
-----

--PROCEDURES FOR CONTROLLING AND READING THE INPUT PORTS

-- Name : ClearSensor
-- Description :
-- This procedure takes in a sensor port and clears the
-- current value of that sensor to '0'. This only needs to
-- be performed on sensors configured as Rotation, Edge, or Pulse,
-- since they are the only sensors that maintain a value.

-- Example: Clear_Sensor(Sensor => Sensor_1);
procedure Clear_Sensor (
    Sensor : in Sensor_Port );

-- Name : Config_Sensor
-- Description :
-- This procedure takes in a sensor type and a config type.
-- It tells the RCX how to configure the given input. It
-- differs from setsensortype because setsensortype deals
-- with the attached hardware and not the output. For
-- example, a touch sensor (hardware) can be used as a touch
-- sensor giving a value of 0 or 1, or as a pulse sensor
-- that keeps a count of presses, or as an edge sensor that
-- keeps a count of state transitions. Depending on how you
-- want to use the sensor, is how you configure it.

```

```

-- Example:  Config_Sensor(Sensor => Sensor_2,
--                   Config => Touch);

procedure Config_Sensor (
    Sensor : in Sensor_Port;
    Config : in Configuration );

-- Name : Get_Sensor_Value
-- Description :
-- Takes in a sensor port and returns the current value.

-- Example:  if (Get_Sensor_Value(Sensor => Sensor_1) = 0) then ...

function Get_Sensor_Value (Sensor : in Sensor_Port )
    return sensor_value;

-- Name : Set_Sensor_Mode
-- Description:
-- This procedure takes in a sensor port and a mode, setting
-- the sensor appropriately.  Normally not used, Config_Sensor
-- is better.

-- Example:  Set_Sensor_Mode(Sensor => Sensor_1, Mode => Pulse);

procedure Set_Sensor_Mode (
    Sensor : in Sensor_Port;
    Mode : in sensor_mode );

-- Name : Set_Sensor_Type
-- Description:
-- This procedure sets the type of a sensor.  Normally not used,
-- Config_Sensor is better.

-- Example:  Set_Sensor_Type(Sensor => Sensor_1, Kind =>
-- Type_Touch);

procedure Set_Sensor_Type(
    Sensor : in Sensor_Port;
    Kind : in Sensor_Type );

-- PROCEDURES FOR CONTROLLING THE OUTPUT PORTS

-- Name : Output_float
-- Description :
-- This procedure takes in an output port (the ports most
-- commonly used to run motors) and turns it off, but in a
-- way that allows it to glide to a stop.

-- Example:  Output_Float(Output => Output_1);

procedure Output_Float (Output : in Output_Port );

```

```

-- Name : Output_Forward
-- Description :
-- This procedure takes in an output port and sets it in the
-- forward direction. The port must be on for this to work.

-- Example: Output_Forward(Output => Output_1);

procedure Output_Forward (Output : in Output_Port );

-- Name : Output_Off
-- Description :
-- This procedure takes in an output port (the ports most
-- commonly used to run motors) and turns it off.

-- Example: Output_Off(Output => Output_1);

procedure Output_Off (Output : in Output_Port );

-- Name : Output_On
-- Description :
-- This procedure takes in an output port (the ports most
-- commonly used to run motors) and turns it on.

-- Example: Output_On(Output => Output_1);

procedure Output_On (Output : in Output_Port );

-- Name : Output_on_for
-- Description :
-- This procedure takes in an output port and a time in hundredths
-- of a second. It turns the output port on for the specified
-- time duration.

-- Example: Output_On_For(Output => Output_1,
--           Hundredths_Of_A_Second => 200);

procedure Output_On_For (
    Output : in Output_Port;
    Hundredths_Of_A_Second : in Natural );

-- Name : Output_on_forward
-- Description :
-- This procedure takes in an output port. It simply turns
-- on the port AND sets it forward.

-- Example: Output_On_Forward(Output => Output_1);

procedure Output_On_Forward (
    Output : in Output_Port );

-- Name : Output_on_reverse
-- Description :
-- This procedure takes in an output port. It simply turns

```

```

-- on the port AND sets it to reverse.

-- Example:  Output_On_Reverse(Output => Output_1)

procedure Output_On_Reverse (
    Output : in Output_Port );

-- Name : Ouput_power
-- Description :
-- This procedure takes in an output port and a power value.
-- It applies this power to the output.  The minimum power
-- is zero (Power_Low) and the max is seven (Power_High).

-- Example:  Output_Power(Output => Output_1, Power => Power_Low);
--           Output_Power(Output => Output_1, Power => 3);

procedure Output_Power (
    Output : in Output_Port;
    Power  : in Power_Type  );

-- Name : Output_Reverse
-- Description :
-- This procedure takes in an output port and sets it in the
-- reverse direction.  The port must be on for this to work.

-- Example:  Output_Reverse(Output => Output_1);

procedure Output_Reverse (Output : in Output_Port );

-- Name : Output_toggle
-- Description :
-- This procedure takes in an output port and flips the
-- direction of the output.  The port must be on for this to
-- work.

-- Example:  Output_Toggle(Output => Output_1);

procedure Output_Toggle (Output : in Output_Port );

-- Name : Set_Output_Direction
-- Description:
-- Sets the direction of a given output.

-- Example:  Set_Output_Direction(Output => Output_1,
--                               Direction => Output_Direction_Forward);

procedure Set_Output_Direction(
    Output : in Output_Port;
    Direction : in Output_Direction);

-- Name : Set_Output_Mode
-- Description:
-- Turns a given output on or off.

```

```

-- Example:  Set_Output_Mode(Output => Output_1, Mode => Output_Mode_On);

procedure Set_Output_Mode(
    Output : in Output_Port;
    Mode : in Output_Mode);

--OTHER PROCEDURES

-- Name : Clear_Message
-- Description :
-- This procedure clears the IR message buffer.  For example,
-- if an RCX receives a message from another RCX, this
-- message is retained until cleared.

-- Example:  Clear_Message;

procedure Clear_Message;

-- Name : Play_Sound
-- Description :  This procedure takes in a sound type and plays it.

-- Example:  Play_Sound(Sound_To_Play => Double_Beep);

procedure Play_Sound (
    Sound_To_Play : in sound );

-- Name : Play_Tone
-- Description :
-- This procedure takes in a frequency and a duration and
-- plays the frequency for that duration.  The frequency
-- can be any in the range of the human ear, and the duration
-- is given in 10ths of a second.

-- Example:  Play_Tone(Frequency_In_Hertz => 440,
--                    Tenths_Of_A_Second => 20);

procedure Play_Tone (
    Frequency_In_Hertz : in frequency;
    Tenths_Of_A_Second : in Natural );

-- Name : Send_Message
-- Description :
-- This procedure takes in a message type.  RCX will send the
-- message (a number from 1 to 255) through its IR port.  This is
-- commonly used for communicating between RCX's.

-- Example:  Send_Message(Message => 124);

procedure Send_Message (
    Message_To_Send : in Message );

-- Name : Stop_All_Tasks

```

```
-- Description : This procedure will stop everything running on the RCX.
-- Example: Stop_All_Tasks;

procedure Stop_All_Tasks;

-- Name : Wait
-- Description :
-- This procedure takes in a time duration and waits for that
-- amount of time before moving on to the next command. The
-- duration is in 100ths of a second.

-- Example: Wait (Hundredths_Of_A_Second => 50);

procedure Wait (Hundredths_Of_A_Second : in Natural );

end Lego;
```