

# Remote I/O Package Specifications

## Introduction

This document contains selected package specifications from `libsimpleio/ada/` that are pertinent to the Ada-Europe 2019 Remote I/O Tutorial.

## Table of Contents

Remote I/O Package Specifications.....	1
Introduction.....	1
Messaging.Fixed.....	2
Message64.....	2
HID.hidapi.....	3
RemoteIO.Client.....	5
RemoteIO.LPC1114.....	6
IO_Interfaces.....	10
GPIO.....	11
GPIO.RemoteIO.....	12
Analog.....	13
Voltage.....	14
ADC.....	15
ADC.RemoteIO.....	16
Servo.....	17
Servo.PWM_Template.....	18
Servo.PWM.....	18

## **Messaging.Fixed**

This generic package defines an abstract interface for fixed length messaging services. It must be instantiated with a message size parameter. It defines an interface type **MessengerInterface** and a classwide access type **Messenger**. It also defines **Byte** and **Message** types.

GENERIC

```
    MessageSize : Natural;
```

```
PACKAGE Messaging.Fixed IS
```

```
    TYPE MessengerInterface IS INTERFACE;
```

```
    TYPE Messenger IS ACCESS ALL MessengerInterface'Class;
```

```
    TYPE Byte IS MOD 256;
```

```
    TYPE Message IS ARRAY (Natural RANGE 0 .. MessageSize -1) OF Byte;
```

```
-- Send a message
```

```
PROCEDURE Send
```

```
    (Self      : MessengerInterface;  
     msg       : IN Message) IS ABSTRACT;
```

```
-- Receive a message
```

```
PROCEDURE Receive
```

```
    (Self      : MessengerInterface;  
     msg       : OUT Message) IS ABSTRACT;
```

```
-- Dump a message in hexadecimal format
```

```
PROCEDURE Dump(msg : Message);
```

```
END Messaging.Fixed;
```

## **Message64**

This package is an instantiation of **Messaging.Fixed** for 64-byte messages. It defines the 64-byte message API used by the Remote I/O Protocol. Packages such as **HID.hidapi** and **Message64.UDP** implement **Message64.MessengerInterface** to provide concrete messaging services.

```
WITH Messaging.Fixed;
```

```
PACKAGE Message64 IS NEW Messaging.Fixed(64);
```

## HID.hidapi

This package implements `Message64.MessengerInterface` as the type `MessengerSubclass`, using the *HIDAPI Library* (<https://github.com/signal11/hidapi>) for communicating with USB raw HID devices.

The `Create` function returns a messenger instance of type `Message64.Messenger`. Optional vendor ID, product ID, and serial number parameters to `Create` can select a specific USB raw HID device if more than one are attached to the host computer.

```
WITH HID.Munts;
WITH Message64;

PRIVATE WITH Interfaces.C;
PRIVATE WITH System;

PACKAGE HID.hidapi IS

    -- Type definitions

    TYPE MessengerSubclass IS NEW Message64.MessengerInterface WITH PRIVATE;

    Destroyed : CONSTANT MessengerSubclass;

    -- Constructor

    -- Allowed values for the timeout parameter:
    --
    -- -1 => Receive operation blocks forever, until a report is received
    --  0 => Receive operation never blocks at all
    -- >0 => Receive operation blocks for the indicated number of milliseconds

    FUNCTION Create
        (vid      : HID.Vendor   := HID.Munts.VID;
         pid      : HID.Product  := HID.Munts.PID;
         serial    : String      := "";
         timeoutms : Integer := 1000) RETURN Message64.Messenger;

    -- Initializer

    PROCEDURE Initialize
        (Self      : IN OUT MessengerSubclass;
         vid      : HID.Vendor   := HID.Munts.VID;
         pid      : HID.Product  := HID.Munts.PID;
         serial    : String      := "";
         timeoutms : Integer := 1000);

    -- Destroyer

    PROCEDURE Destroy(Self : IN OUT MessengerSubclass);

    -- Send a message

    PROCEDURE Send
        (Self : MessengerSubclass;
         msg  : Message64.Message);
```

```

-- Receive a message

PROCEDURE Receive
  (Self : MessengerSubclass;
   msg  : OUT Message64.Message);

-- Get HID device name string

FUNCTION Name(Self : MessengerSubclass) RETURN String;

-- Get HID device manufacturer string

FUNCTION Manufacturer
  (Self : MessengerSubclass) RETURN String;

-- Get HID device product string

FUNCTION Product
  (Self : MessengerSubclass) RETURN String;

-- Get HID device serial number string

FUNCTION SerialNumber
  (Self : MessengerSubclass) RETURN String;

PRIVATE
  -- Implementation defined
END HID.hidapi;

```

## RemoteIO.Client

This package defines the client API for the Remote I/O Protocol. It defines a class type `DeviceClass` and a classwide access type `Device`. The `Create` function accepts an object instance of type `Message64.Messenger` (e.g. from `HID.hidapi.Create`) and returns an object instance of type `Device`. As usual, `Create` is *not* a primitive operation of `DeviceClass`. The four methods `Transaction`, `GetVersion`, `GetCapability`, and `GetAvailableChannels` *are* primitive operations of `DeviceClass`.

```
WITH Message64;
```

```
PACKAGE RemoteIO.Client IS
```

```
-- Define a tagged type for remote I/O server devices
```

```
TYPE DeviceClass IS TAGGED PRIVATE;
```

```
-- Define an access type compatible with any subclass implementing  
-- DeviceClass
```

```
TYPE Device IS ACCESS ALL DeviceClass'Class;
```

```
-- Constructors
```

```
FUNCTION Create(msg : Message64.Messenger) RETURN Device;
```

```
-- Perform a Remote I/O operation
```

```
PROCEDURE Transaction
```

```
(Self : IN OUT DeviceClass;  
 cmd  : IN OUT Message64.Message;  
 resp : OUT Message64.Message);
```

```
-- Get the remote device version string
```

```
FUNCTION GetVersion(Self : IN OUT DeviceClass) RETURN String;
```

```
-- Get the remote device capability string
```

```
FUNCTION GetCapability(Self : IN OUT DeviceClass) RETURN String;
```

```
-- Get the available channels for a given service type
```

```
FUNCTION GetAvailableChannels
```

```
(Self      : IN OUT DeviceClass;  
 service   : ChannelTypes) RETURN ChannelSets.Set;
```

```
PRIVATE
```

```
-- Implementation defined
```

```
END RemoteIO.Client;
```

## RemoteIO.LPC1114

This package specifies all of the resources available from the LPC1114 I/O Processor Remote I/O Server. Packages for other Remote I/O server devices (e.g. `RemoteIO.FEZ`) are available in `libsimpleio/ada/remoteio/client/`.

The first section (this page) defines constants for the normal Remote I/O resources (analog inputs, GPIO pins, and PWM outputs).

The second section (the three following pages) defines a Remote I/O abstract device binding to the SPI Agent firmware inside the LPC1114 I/O Processor. See the ***Raspberry Pi LPC1114 I/O Processor Expansion Board User Guide*** for more information about the services the SPI Agent firmware can provide:

<http://git.munts.com/rpi-mcu/expansion/LPC1114/doc/UserGuide.pdf>

```
WITH Interfaces; USE Interfaces;
```

```
WITH Message64;
```

```
WITH RemoteIO.Abstract_Device;
```

```
WITH RemoteIO.Client;
```

```
PACKAGE RemoteIO.LPC1114 IS
```

```
-- Analog inputs
```

```
AIN1  : CONSTANT RemoteIO.ChannelNumber := 1;  -- aka LPC1114 P1.0
AIN2  : CONSTANT RemoteIO.ChannelNumber := 2;  -- aka LPC1114 P1.1
AIN3  : CONSTANT RemoteIO.ChannelNumber := 3;  -- aka LPC1114 P1.2
AIN4  : CONSTANT RemoteIO.ChannelNumber := 4;  -- aka LPC1114 P1.3
AIN5  : CONSTANT RemoteIO.ChannelNumber := 5;  -- aka LPC1114 P1.4
```

```
-- GPIO pins
```

```
LED   : CONSTANT RemoteIO.ChannelNumber := 0;  -- aka LPC1114 P0.7
GPIO0 : CONSTANT RemoteIO.ChannelNumber := 1;  -- aka LPC1114 P1.0
GPIO1 : CONSTANT RemoteIO.ChannelNumber := 2;  -- aka LPC1114 P1.1
GPIO2 : CONSTANT RemoteIO.ChannelNumber := 3;  -- aka LPC1114 P1.2
GPIO3 : CONSTANT RemoteIO.ChannelNumber := 4;  -- aka LPC1114 P1.3
GPIO4 : CONSTANT RemoteIO.ChannelNumber := 5;  -- aka LPC1114 P1.4
GPIO5 : CONSTANT RemoteIO.ChannelNumber := 6;  -- aka LPC1114 P1.5
GPIO6 : CONSTANT RemoteIO.ChannelNumber := 7;  -- aka LPC1114 P1.8
GPIO7 : CONSTANT RemoteIO.ChannelNumber := 8;  -- aka LPC1114 P1.9
```

```
-- PWM outputs
```

```
PWM1  : CONSTANT RemoteIO.ChannelNumber := 1;  -- aka LPC1114 P1.1
PWM2  : CONSTANT RemoteIO.ChannelNumber := 2;  -- aka LPC1114 P1.2
PWM3  : CONSTANT RemoteIO.ChannelNumber := 3;  -- aka LPC1114 P1.3
PWM4  : CONSTANT RemoteIO.ChannelNumber := 4;  -- aka LPC1114 P1.9
```

```

-----
--
-- LPC1114 I/O Processor Expansion Board Abstract Device services follow
--
-- See: http://git.munts.com/rpi-mcu/expansion/LPC1114/doc/UserGuide.pdf
--
-- Naming of identifiers below matches UserGuide.pdf.
--
-----

-- Raspberry Pi LPC1114 I/O Processor SPI Agent Firmware command structure

TYPE SPIAGENT_COMMAND_MSG_t IS RECORD
  command : Unsigned_32;
  pin      : Unsigned_32;
  data     : Unsigned_32;
END record;

-- Raspberry Pi LPC1114 I/O Processor SPI Agent Firmware response structure

TYPE SPIAGENT_RESPONSE_MSG_t IS RECORD
  command : Unsigned_32;
  pin      : Unsigned_32;
  data     : Unsigned_32;
  error    : Unsigned_32;
END record;

-- Raspberry Pi LPC1114 I/O Processor SPI Agent Firmware commands

SPIAGENT_CMD_NOP : CONSTANT Unsigned_32 := 0;
SPIAGENT_CMD_LOOPBACK : CONSTANT Unsigned_32 := 1;
SPIAGENT_CMD_CONFIGURE_ANALOG_INPUT : CONSTANT Unsigned_32 := 2;
SPIAGENT_CMD_CONFIGURE_GPIO_INPUT : CONSTANT Unsigned_32 := 3;
SPIAGENT_CMD_CONFIGURE_GPIO_OUTPUT : CONSTANT Unsigned_32 := 4;
SPIAGENT_CMD_CONFIGURE_PWM_OUTPUT : CONSTANT Unsigned_32 := 5;
SPIAGENT_CMD_GET_ANALOG : CONSTANT Unsigned_32 := 6;
SPIAGENT_CMD_GET_GPIO : CONSTANT Unsigned_32 := 7;
SPIAGENT_CMD_PUT_GPIO : CONSTANT Unsigned_32 := 8;
SPIAGENT_CMD_PUT_PWM : CONSTANT Unsigned_32 := 9;
SPIAGENT_CMD_CONFIGURE_GPIO_INTERRUPT : CONSTANT Unsigned_32 := 10;
SPIAGENT_CMD_CONFIGURE_GPIO : CONSTANT Unsigned_32 := 11;
SPIAGENT_CMD_PUT_LEGORC : CONSTANT Unsigned_32 := 12;
SPIAGENT_CMD_GET_SFR : CONSTANT Unsigned_32 := 13;
SPIAGENT_CMD_PUT_SFR : CONSTANT Unsigned_32 := 14;
SPIAGENT_CMD_CONFIGURE_TIMER_MODE : CONSTANT Unsigned_32 := 15;
SPIAGENT_CMD_CONFIGURE_TIMER_PRESCALER : CONSTANT Unsigned_32 := 16;
SPIAGENT_CMD_CONFIGURE_TIMER_CAPTURE : CONSTANT Unsigned_32 := 17;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH0 : CONSTANT Unsigned_32 := 18;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH1 : CONSTANT Unsigned_32 := 19;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH2 : CONSTANT Unsigned_32 := 20;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH3 : CONSTANT Unsigned_32 := 21;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH0_VALUE : CONSTANT Unsigned_32 := 22;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH1_VALUE : CONSTANT Unsigned_32 := 23;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH2_VALUE : CONSTANT Unsigned_32 := 24;
SPIAGENT_CMD_CONFIGURE_TIMER_MATCH3_VALUE : CONSTANT Unsigned_32 := 25;
SPIAGENT_CMD_GET_TIMER_VALUE : CONSTANT Unsigned_32 := 26;
SPIAGENT_CMD_GET_TIMER_CAPTURE : CONSTANT Unsigned_32 := 27;
SPIAGENT_CMD_GET_TIMER_CAPTURE_DELTA : CONSTANT Unsigned_32 := 28;
SPIAGENT_CMD_INIT_TIMER : CONSTANT Unsigned_32 := 29;

```

-- Raspberry Pi LPC1114 I/O Processor General Purpose Input/Output pins

```
LPC1114_GPIO0      : CONSTANT Unsigned_32 := 12; -- aka P1.0
LPC1114_GPIO1      : CONSTANT Unsigned_32 := 13; -- aka P1.1
LPC1114_GPIO2      : CONSTANT Unsigned_32 := 14; -- aka P1.2
LPC1114_GPIO3      : CONSTANT Unsigned_32 := 15; -- aka P1.3
LPC1114_GPIO4      : CONSTANT Unsigned_32 := 16; -- aka P1.4
LPC1114_GPIO5      : CONSTANT Unsigned_32 := 17; -- aka P1.5
LPC1114_GPIO6      : CONSTANT Unsigned_32 := 20; -- aka P1.8
LPC1114_GPIO7      : CONSTANT Unsigned_32 := 21; -- aka P1.9
LPC1114_LED        : CONSTANT Unsigned_32 := 7;  -- aka P0.7
```

-- Raspberry Pi LPC1114 I/O Processor Analog input pins

```
LPC1114_AD1        : CONSTANT Unsigned_32 := LPC1114_GPIO0;
LPC1114_AD2        : CONSTANT Unsigned_32 := LPC1114_GPIO1;
LPC1114_AD3        : CONSTANT Unsigned_32 := LPC1114_GPIO2;
LPC1114_AD4        : CONSTANT Unsigned_32 := LPC1114_GPIO3;
LPC1114_AD5        : CONSTANT Unsigned_32 := LPC1114_GPIO4;
```

-- Raspberry Pi LPC1114 I/O Processor PWM output pins

```
LPC1114_PWM1       : CONSTANT Unsigned_32 := LPC1114_GPIO1;
LPC1114_PWM2       : CONSTANT Unsigned_32 := LPC1114_GPIO2;
LPC1114_PWM3       : CONSTANT Unsigned_32 := LPC1114_GPIO3;
LPC1114_PWM4       : CONSTANT Unsigned_32 := LPC1114_GPIO7;
```

-- Raspberry Pi LPC1114 I/O Processor Timer pins

```
LPC1114_CT32B1_CAP0 : CONSTANT Unsigned_32 := LPC1114_GPIO0;
LPC1114_CT32B1_MAT0 : CONSTANT Unsigned_32 := LPC1114_GPIO1;
LPC1114_CT32B1_MAT1 : CONSTANT Unsigned_32 := LPC1114_GPIO2;
LPC1114_CT32B1_MAT2 : CONSTANT Unsigned_32 := LPC1114_GPIO3;
LPC1114_CT32B1_MAT3 : CONSTANT Unsigned_32 := LPC1114_GPIO4;
LPC1114_CT32B0_CAP0 : CONSTANT Unsigned_32 := LPC1114_GPIO5;
```

-- LPC1114 special function registers

```
LPC1114_DEVICEID    : CONSTANT Unsigned_32 := 16#400483F4#;
LPC1114_GPIO1DATA    : CONSTANT Unsigned_32 := 16#50010CFC#;
LPC1114_U0SCR        : CONSTANT Unsigned_32 := 16#4000801C#;
```

-- LPC1114 GPIO pin modes

```
LPC1114_GPIO_MODE_INPUT      : CONSTANT Unsigned_32 := 0; -- High Z input
LPC1114_GPIO_MODE_INPUT_PULLDOWN : CONSTANT Unsigned_32 := 1;
LPC1114_GPIO_MODE_INPUT_PULLUP   : CONSTANT Unsigned_32 := 2;
LPC1114_GPIO_MODE_OUTPUT       : CONSTANT Unsigned_32 := 3; -- Push-pull output
LPC1114_GPIO_MODE_OUTPUT_OPENDRAIN : CONSTANT Unsigned_32 := 4;
```

-- LPC1114 timer identifiers

```
LPC1114_CT32B0      : CONSTANT Unsigned_32 := 0;
LPC1114_CT32B1      : CONSTANT Unsigned_32 := 1;
```

-- LPC1114 timer modes

```
LPC1114_TIMER_MODE_DISABLED : CONSTANT Unsigned_32 := 0;
LPC1114_TIMER_MODE_RESET    : CONSTANT Unsigned_32 := 1;
LPC1114_TIMER_MODE_PCLK     : CONSTANT Unsigned_32 := 2;
LPC1114_TIMER_MODE_CAP0_RISING : CONSTANT Unsigned_32 := 3;
LPC1114_TIMER_MODE_CAP0_FALLING : CONSTANT Unsigned_32 := 4;
LPC1114_TIMER_MODE_CAP0_BOTH  : CONSTANT Unsigned_32 := 5;
```



```

-- LPC1114 timer capture edges

LPC1114_TIMER_CAPTURE_EDGE_DISABLED      : CONSTANT Unsigned_32 := 0;
LPC1114_TIMER_CAPTURE_EDGE_CAP0_RISING   : CONSTANT Unsigned_32 := 1;
LPC1114_TIMER_CAPTURE_EDGE_CAP0_FALLING  : CONSTANT Unsigned_32 := 2;
LPC1114_TIMER_CAPTURE_EDGE_CAP0_BOTH     : CONSTANT Unsigned_32 := 3;

-- LPC1114 timer match registers

LPC1114_TIMER_MATCH0 : CONSTANT Unsigned_32 := 0;
LPC1114_TIMER_MATCH1 : CONSTANT Unsigned_32 := 1;
LPC1114_TIMER_MATCH2 : CONSTANT Unsigned_32 := 2;
LPC1114_TIMER_MATCH3 : CONSTANT Unsigned_32 := 3;

-- LPC1114 timer match output actions

LPC1114_TIMER_MATCH_OUTPUT_DISABLED : CONSTANT Unsigned_32 := 0;
LPC1114_TIMER_MATCH_OUTPUT_CLEAR    : CONSTANT Unsigned_32 := 1;
LPC1114_TIMER_MATCH_OUTPUT_SET       : CONSTANT Unsigned_32 := 2;
LPC1114_TIMER_MATCH_OUTPUT_TOGGLE   : CONSTANT Unsigned_32 := 3;

-- Instantiate RemoteIO.Abstract_Device

FUNCTION FromCommand(cmd : SPIAGENT_COMMAND_MSG_t) RETURN Message64.Message;

FUNCTION ToResponse(msg : Message64.Message) RETURN SPIAGENT_RESPONSE_MSG_t;

PACKAGE Abstract_Device IS NEW RemoteIO.Abstract_Device
  (SPIAGENT_COMMAND_MSG_t, SPIAGENT_RESPONSE_MSG_t);

END RemoteIO.LPC1114;

```

## IO\_Interfaces

This generic package must be instantiated with some type **Property**, which can be scalar or composite. The instantiated package will define three abstract interface types: **InputInterface**, **InputOutputInterface**, and **OutputInterface**. Each abstract interface includes a classwide access type and **Get** and/or **Put** procedures. **IO\_Interfaces** is used extensively internally within **libsimpleio** but will seldom if ever be need to be referenced from an application program.

GENERIC

```
    TYPE Property IS PRIVATE;

PACKAGE IO_Interfaces IS

    -- Define an abstract input only interface

    TYPE InputInterface IS INTERFACE;

    -- Define a method for reading from an input

    FUNCTION Get(Self : IN OUT InputInterface) RETURN Property IS ABSTRACT;

    -----

    -- Define an abstract input/output interface

    TYPE InputOutputInterface IS INTERFACE;

    -- Define a method for reading from an input

    FUNCTION Get(Self : IN OUT InputOutputInterface) RETURN Property IS ABSTRACT;

    -- Define a method for writing to an output

    PROCEDURE Put(Self : IN OUT InputOutputInterface; value : Property) IS ABSTRACT;

    -----

    -- Define an abstract output only interface

    TYPE OutputInterface IS INTERFACE;

    -- Define a method for writing to an output

    PROCEDURE Put(Self : IN OUT OutputInterface; value : Property) IS ABSTRACT;

END IO_Interfaces;
```

## **GPIO**

This package defines an abstract interface for all GPIO (General Purpose Input/Output) pins. It defines an exception `GPIO_Error`, an abstract interface type `PinInterface`, and a classwide access type `Pin`.

```
WITH Ada.Text_IO;
WITH IO_Interfaces;

PACKAGE GPIO IS

    -- Define an exception for GPIO errors

    GPIO_Error : EXCEPTION;

    -- Instantiate text I/O package

    PACKAGE Boolean_IO IS NEW Ada.Text_IO Enumeration_IO (Boolean);

    -- Type definitions

    TYPE Direction IS (Input, Output);

    -- Instantiate I/O interfaces package for digital I/O

    PACKAGE Interfaces IS NEW IO_Interfaces (Boolean);

    -- Define an abstract interface for GPIO pins, derived from
    -- Interfaces.InputOutputInterface

    TYPE PinInterface IS INTERFACE AND Interfaces.InputOutputInterface;

    -- Define an access type compatible with any subclass implementing
    -- PinInterface

    TYPE Pin IS ACCESS ALL PinInterface'Class;

END GPIO;
```

## GPIO.RemoteIO

This package provides GPIO pin services using the Remote I/O protocol. It defines a concrete subclass of `GPIO.PinInterface` called `GPIO.RemoteIO.PinSubclass`.

Note that the `Create` function returns a value of classwide access type `GPIO.Pin` and is ***not*** a primitive operation of `GPIO.RemoteIO.PinSubclass`. This is a pattern followed throughout `libsimpleio`.

```
WITH RemoteIO.Client;

PACKAGE GPIO.RemoteIO IS

    TYPE PinSubclass IS NEW PinInterface WITH PRIVATE;

    -- GPIO pin object constructor

    FUNCTION Create
        (dev    : Standard.RemoteIO.Client.Device;
         num    : Standard.RemoteIO.ChannelNumber;
         dir    : Direction;
         state  : Boolean := False) RETURN Pin;

    -- Read GPIO pin state

    FUNCTION Get(Self : IN OUT PinSubclass) RETURN Boolean;

    -- Write GPIO pin state

    PROCEDURE Put(Self : IN OUT PinSubclass; state : Boolean);

PRIVATE
    -- Implementation defined
END GPIO.RemoteIO;
```

## Analog

This package defines abstract interfaces for analog sampled data inputs, outputs, and input/outputs. Use `InputInterface` and `Input` for ADC (Analog to Digital Converter) inputs and `OutputInterface` and `Output` for DAC (Digital to Analog Converter) outputs.

`InputOutputInterface` and `InputOutput` are provided for completeness. They might be useful for a DAC with readback capability, or for unusual devices that are configurable as either analog input or output.

Sampled analog data values (of type `Sample`) are 32-bit unsigned and right justified.

```
WITH Ada.Text_IO;
WITH IO_Interfaces;

PACKAGE Analog IS

  -- Define a type for sampled analog data

  MaxResolution : CONSTANT := 32; -- Bits

  TYPE Sample IS MOD 2**MaxResolution;

  -- Instantiate text I/O package

  PACKAGE Sample_IO IS NEW Ada.Text_IO.Modular_IO(Sample);

  -- Instantiate abstract interfaces package

  PACKAGE Interfaces IS NEW IO_Interfaces(Sample);

  -- Interfaces

  TYPE InputInterface IS INTERFACE AND Interfaces.InputInterface;

  TYPE OutputInterface IS INTERFACE AND Interfaces.OutputInterface;

  TYPE InputOutputInterface IS INTERFACE AND Interfaces.InputOutputInterface;

  -- Access types

  TYPE Input IS ACCESS ALL InputInterface'Class;

  TYPE Output IS ACCESS ALL OutputInterface'Class;

  TYPE InputOutput IS ACCESS ALL InputOutputInterface'Class;

  -- Additional methods

  FUNCTION GetResolution(Self : IN OUT InputInterface) RETURN Positive IS ABSTRACT;

  FUNCTION GetResolution(Self : IN OUT InputOutputInterface) RETURN Positive IS
    ABSTRACT;

  FUNCTION GetResolution(Self : IN OUT OutputInterface) RETURN Positive IS ABSTRACT;

END Analog;
```

## Voltage

This package defines a type `Volts` to represent continuously variable voltage input and/or output devices. It is representative of all of the physical quantity packages.

```
WITH Ada.Text_IO;
WITH IO_Interfaces;

PACKAGE Voltage IS

    TYPE Volts IS NEW Float;

    -- Instantiate text I/O package

    PACKAGE Volts_IO IS NEW Ada.Text_IO.Float_IO(Volts);

    -- Instantiate abstract interfaces package

    PACKAGE Interfaces IS NEW IO_Interfaces(Volts);

    -- Interfaces

    TYPE InputInterface IS INTERFACE AND Interfaces.InputInterface;

    TYPE OutputInterface IS INTERFACE AND Interfaces.OutputInterface;

    TYPE InputOutputInterface IS INTERFACE AND Interfaces.InputOutputInterface;

    -- Access types

    TYPE Input IS ACCESS ALL InputInterface'Class;

    TYPE Output IS ACCESS ALL OutputInterface'Class;

    TYPE InputOutput IS ACCESS ALL InputOutputInterface'Class;

END Voltage;
```

## **ADC**

This package provides services for reading the scaled input voltage from ADC (Analog to Digital Converter) inputs. It defines a concrete subclass of `Voltage.Interfaces.InputInterface` called `ADC.InputSubclass`.

The `Create` function accepts an analog input object instance (of type `Analog.Input`), a reference voltage value (of type `Voltage.Volts`), and a voltage gain value (also of type `Voltage.Volts`) and returns a voltage input object instance of type `Voltage.Interfaces.Input`.

As is usual throughout `libsimpleio`, `Create` is **not** a primitive operation of `ADC.InputSubclass`.

```
WITH Analog;
WITH Voltage;

PACKAGE ADC IS

    ADC_Error : EXCEPTION;

    TYPE InputSubclass IS NEW Voltage.InputInterface WITH PRIVATE;

    -- Constructor

    FUNCTION Create
        (input      : Analog.Input;
         reference  : Voltage.Volts;
         gain       : Voltage.Volts := 1.0) RETURN Voltage.Input;

    -- Methods

    FUNCTION Get(Self : IN OUT InputSubclass) RETURN Voltage.Volts;

PRIVATE
    -- Implementation defined
END ADC;
```

## **ADC.RemoteIO**

This package provides analog input services using the Remote I/O protocol. It defines a concrete subclass of `Analog.InputInterface` called `ADC.RemoteIO.InputSubclass`.

The `Create` function returns an analog input object instance of type `Analog.Input`.

```
WITH Analog;
WITH RemoteIO.Client;

PACKAGE ADC.RemoteIO IS

    TYPE InputSubclass IS NEW Analog.InputInterface WITH PRIVATE;

    -- A/D input pin object constructor

    FUNCTION Create
        (dev : Standard.RemoteIO.Client.Device;
         num : Standard.RemoteIO.ChannelNumber) RETURN Analog.Input;

    -- Read A/D input pin

    FUNCTION Get(Self : IN OUT InputSubclass) RETURN Analog.Sample;

    -- Retrieve A/D input resolution

    FUNCTION GetResolution(Self : IN OUT InputSubclass) RETURN Positive;

PRIVATE
    -- Implementation defined
END ADC.RemoteIO;
```



## Servo

This package defines an abstract interface for all RC (Remote Control) servo motor outputs. Servo motors have been used for many years in model airplanes and boats for mechanically controlling flaps, rudder, and throttle. They are now also commonly used in robotics.

```
WITH Ada.Text_IO;
WITH IO_Interfaces;

PACKAGE Servo IS

    Servo_Error : EXCEPTION;

    TYPE Position IS NEW Float RANGE -1.0 .. 1.0;

    MinimumPosition : CONSTANT Position := Position'First;
    NeutralPosition  : CONSTANT Position := 0.0;
    MaximumPosition  : CONSTANT Position := Position'Last;

    -- Instantiate text I/O package

    PACKAGE Position_IO IS NEW Ada.Text_IO.Float_IO(Position);

    -- Instantiate abstract interfaces package

    PACKAGE Interfaces IS NEW IO_Interfaces(Position);

    -- Define an abstract interface for servo outputs, derived from
    -- Interfaces.OutputInterface

    TYPE OutputInterface IS INTERFACE AND Interfaces.OutputInterface;

    -- Define an access type compatible with any subclass implementing
    -- OutputInterface

    TYPE Output IS ACCESS ALL OutputInterface'Class;

END Servo;
```

## Servo.PWM\_Template

The pulse train for a servo motor can be implemented using a PWM (Pulse Width Modulator) device, by constraining the frequency and width of the PWM pulse train to what is required by a particular servo motor.

This generic package must be instantiated with two pulse width values, for the minimum and maximum pulse widths supported by a particular servo motor. The constructor requires a PWM output object instance of type **PWM.Output** (along with an optional position parameter) and returns a servo output object instance of type **Servo.Output**.

```
WITH PWM;
```

```
GENERIC
```

```
    MinimumWidth : IN Duration;  
    MaximumWidth : IN Duration;
```

```
PACKAGE Servo.PWM_Template IS
```

```
-- Type definitions
```

```
TYPE OutputSubclass IS NEW Servo.OutputInterface WITH PRIVATE;
```

```
-- Servo output object constructor
```

```
FUNCTION Create
```

```
    (output      : PWM.Output;  
     position    : Servo.Position := Servo.NeutralPosition)  
    RETURN Servo.Output;
```

```
-- Servo output write method
```

```
PROCEDURE Put
```

```
    (Self        : IN OUT OutputSubclass;  
     position    : Servo.Position);
```

```
PRIVATE
```

```
    - Implementation defined
```

```
END Servo.PWM_Template;
```

## Servo.PWM

Conventional hobby RC servo motors require a 50 Hz pulse train with pulse widths ranging from 1.0 to 2.0 milliseconds. A pulse width of 1.0 milliseconds selects one extreme position and 2.0 milliseconds selects the other. A pulse width of 1.5 milliseconds places the servo into the center or neutral position. This package is an instantiation of **Servo.PWM\_Template** for such servos.

```
WITH Servo.PWM_Template;
```

```
PACKAGE Servo.PWM IS NEW Servo.PWM_Template(1.0E-3, 2.0E-3);
```