**6 XDR2OS3**

**6.1 Requirements**

**6.1.1 Introduction**

XDR2OS3 is a tool that comes open source with CubedOS, a modular application framework for small satellites (CubeSats). XDR2OS3 is written in Scala and takes as user input a subset of the speciﬁed formal language of XDR (with some modiﬁcation) and produces provable SPARK 2014 as output (\*.ads and concomitant \*.adb). The user may then manually insert the code into their project and test it. The SPARK 2014 code generated by XDR2OS3 shall be the procedures and functions used to encode and decode CubedOS messages for inter-modular communications--repetitive, boilerplate-like code.

**6.1.2 Key Components**

**SCALA**

Scala is a general-purpose-language running on the Java Virtual Machine (JVM) that is a Frankenstein mixture of functional and imperative programming. Here it is being used to create the ANTLR-based compiler tool XDR2OS3.

**GRAMMAR**

A grammar is an arbitrary set of rules, each one expressing the structure of a phrase. ANTLR translates a grammar to a parser. See Appendix A for full .g4 grammar file.

**LEXER**

The lexer is an ANTLR generated file custom designed to perform lexical analysis (or tokenizing) on the given input. The lexer groups related tokens into token types (INT, ID, STRUCT, etc.) A token consists of two pieces of information: token type and the text matched for the token by the lexer.

**PARSER**

The parser is also an ANTLR generated file, based off of a supplied grammar, that cares only about the type of an individual token. The parser uses a parse tree (or syntax tree) to recognize the structure of the input sentence and its component phrases. ANTLR parsers descend recursively, beginning at the root of the parse tree and proceeding toward the leaves (tokens).

**PARSE TREE**

Generated from the parser, the parse tree is easy to process in subsequent steps and is reusable. The inner nodes of the parse tree are phrase names that group and identify their children, with the root node being the most abstract phrase name. The leaves of a parse tree are the input tokens.

**SEMANTIC ANALYZER**

The semantic analyzer is a file intended to ensure that the meaning of the sentence structure is correct. Whereas the other files are concerned with the correctness of the sentence syntax and the correct structure of the generated parse tree, the semantic analyzer makes sure that the sentence structure makes sense from the point-of-view of the XDR2OS3 user.

**SYMBOL TABLE AND CONCOMITTANT FILES**

The symbol table is the first file invoked by main. It makes sure that no IDENTIFIERs are any variation of SPARK Ada reserved words and then populates several Scala maps for later use in the Specification Generator and Body Generator. There are 3 Scala maps: one that keeps track of structured types such as message structs, structs and enums, one that keeps track of unstructured types such as anything else, and one that keeps track of the element members of some structured types such as the message struct and struct. Several methods are frequently invoked in the code generation files to retrieve and manipulate information contained in these maps.

**MXDR**

MXDR is based off of XDR, a C-derived Data Representation Language similar to XML and designed to move the data itself from one ‘node’ to another.

XDR2OS3 shall modify the XDR specification to include ranges of types. E.g. typedef:

type-def:

"typedef" declaration ";"

shall optionally become:

type-def:

"typedef" declaration “range” range\_constraint ";"

where range\_constraint is defined as:

range\_constraint:

CONSTANT DOTDOT CONSTANT

| CONSTANT DOTDOT IDENTIFIER;(see Appendix A for full .g4 file)

XDR2OS3 shall incorporate the “message struct” which is split by XDR2OS3 into a decode procedure, an encode function and a message “check” that verifies the message based on its sender parameter and the type of message as proscribed in its message ID.

E.g.:

message struct -> Example {

Width\_Type thing;

};

shall produce:

function Example\_Encode

(Sender\_Domain : Domain\_ID\_Type;

Sender : Module\_ID\_Type;

thing : Width\_Type;

Priority : System.Priority := System.Default\_Priority) return Message\_Record

with

Global => null;

function Is\_Example(Message : Message\_Record) return Boolean is

(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Example));

procedure Example\_Decode

(Message : in Message\_Record;

thing : out Width\_Type;

Decode\_Status : out Message\_Status\_Type)

with

Global => null,

Pre => Is\_Example(Message),

Depends => ((thing, Decode\_Status) => Message);

in the SPARK \*.ads file.

In the SPARK \*.adb file:

function Example\_Encode

(Sender\_Domain : Domain\_ID\_Type;

Sender : Module\_ID\_Type;

thing : Width\_Type;

Priority : System.Priority := System.Default\_Priority) return Message\_Record

is

Message : Message\_Record := Make\_Empty\_Message(

Sender\_Domain => Sender\_Domain,

Receiver\_Domain => Domain\_ID,

Sender => Sender,

Receiver => ID,

Message\_ID => Message\_Type'Pos(Example),

Priority => Priority);

Position : XDR\_Index\_Type;

Last : XDR\_Index\_Type;

begin

Position := 0;

XDR.Encode(XDR.XDR\_Unsigned(thing), Message.Payload, Position, Last);

Position := Last + 1;

Message.Size := Position;

return Message;

end Example\_Encode;

procedure Example\_Decode

(Message : in Message\_Record;

thing : out Width\_Type;

Decode\_Status : out Message\_Status\_Type)

is

Position : XDR\_Index\_Type;

Raw\_thing : XDR.XDR\_Unsigned;

Last : XDR\_Index\_Type;

begin

Decode\_Status := Success;

thing := Width\_Type'First;

Position := 0;

if Decode\_Status = Success then

XDR.Decode(Message.Payload, Position, Raw\_thing, Last);

Position := Last + 1;

if Raw\_thing in XDR.XDR\_Unsigned(Width\_Type'First) .. XDR.XDR\_Unsigned(Width\_Type'Last) then

thing := Width\_Type(Raw\_thing);

Decode\_Status := Success;

else

Decode\_Status := Malformed;

end if;

end if;

end Example\_Decode;

**SPARK**

SPARK 2014 is a subset language of Ada 2012 released April 30, 2014. This High Integrity Language incorporates parts of the Ada 2012 language that can be run through certain tools built into SPARK IDEs that run mathematical calculations on the provided code to ensure that the code meets the design objectives of SPARK: logical soundness, semantic soundness, security, verifiability, bounded resource requirements, and runtime system requirements. The other design objective of SPARK 2014 was to find a meeting point between maintaining rigorous formal definition while not sacrificing its expressive and artistic power. The first version of SPARK was designed by University of Southampton Bernard Carre and Trevor Jennings based on the original Ada ’83. It has subsequently been revamped typically 2 to 3 years after each of Ada’s revampings.

With SPARK 2014, contracts were added to the code to improve tools such as GNATprove in determining the designer’s intentions by comparing the contract along with its pre, post and verification conditions against what the code is doing. This makes for the prevention of the hypothetical scenario where, even though the code is doing something legal and correct, it still wasn’t the designer’s intention. Also, contracts help for an added layer of security and verifiability in general and can help the designer through the coding process.

Here is an example of a SPARK procedure with a contract comprised of one precondition and one postcondition:

procedure Locate\_Node(Position : out Node\_Index)

with Pre => not isFull,

Post => Node\_Array(Position).used = False

is

begin…

Industrially, SPARK 2014 is currently ubiquitous, especially in the security and safety realms. Vermont Technical College (VTC) is currently using SPARK 2014 on its CubedOS application framework.

The most important line of any SPARK program in order that it be differentiated from original Ada:

pragma SPARK\_Mode(On);

This one-liner designates that what follows, along with the file’s associated files (the specification file that defines public and private functions and procedures, the body file that holds the complete functions and procedures of the project itself, and the main file that manipulates public functions and procedures in the specification file), is in fact SPARK and not full Ada.

What exactly separates Ada from SPARK? SPARK has no access types such as pointers, no unstructured control flow such as goto’s, no exception handling, no aliasing of outputs from subprograms, and no side-effects in expressions and functions. All these “normal things” found in many other languages make the formal reasoning and mathematical verifications much harder, at very least, if not impossible.

**MESSAGE INVARIANTS**

A message invariant represents a condition checked either before or after (or both) the execution of a function or procedure in SPARK 2014. An infinite number of message\_invariants (Pre and Post conditions on specification subprograms) may be added to the end of the message struct:

message struct <- Read\_Reply {

Valid\_File\_Handle\_Type Handle;

Read\_Result\_Size\_Type Amount;

opaque Message\_Data<>;

} with message\_invariant => Amount <= Message\_Data'Length,

with message\_invariant => “another message invariant”;

In this example, the “<-“ symbol indicates that the subprograms specified by this input shall be sending rather than receiving the specified type of message. “Handle” and “Amount” are previously defined types and “Message\_Data” was inserted without being previously defined in the MXDR file. The message\_invariants are specified after the message struct body, separated by commas. Currently, only Pre and Post conditions of simple equality are valid: ‘<’, ‘>’, ‘<=’, ‘>=’, ‘=’. (see Appendix A for full .g4 file)

**SPARK SKELETONS**

A generic SPARK template with exactly 3 types of subprograms per message type to be auto-generated into the file. The body template would result in 2 types of subprograms per message type.

--------------------------------------------------------------------------------

-- FILE : cubedos-%MODULENAME%-api.ads

-- SUBJECT: Specification of a package that defines the %MODULENAME% API

-- AUTHOR : (C) Copyright 2017 by Vermont Technical College

-- All the subprograms in this package must be task safe. They can be simultaneously

--called from multiple tasks. If possible, make every subprogram here a pure function.

--------------------------------------------------------------------------------

with CubedOS.Lib;

with Message\_Manager; use Message\_Manager;

with System;

package CubedOS.%MODULENAME%.API is

%BULK%

end CubedOS.%MODULENAME%.API;

**ARRAYS AND PREVIOUSLY DEFINED IDENTIFIERS**

Arrays may be defined per section 6.2.4. Additionally, previously defined identifiers associated with a built-in type may be used as the type of a new identifier.

typedef int I range 1 .. 200;

typedef I Y[100];

In this example, we define a type I as an int within a specified range. Y would then be an array of 100 Is. We can then use this new type in a message struct:

message struct -> Message\_Struct {

Y indexes;

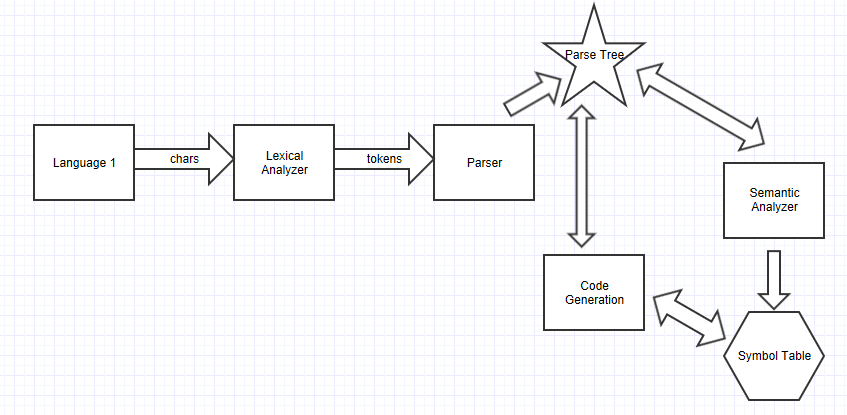
};

**6.1.3 Behavior Characteristics**

XDR2OS3 must eventually be robust enough to handle all conceivable forms of human error and output a message aimed at the elucidation of the user in each case of error, no matter how obscure or intentional. XDR2OS3 must be simple enough, both in user learning and understanding of the tool and its use in functionality and use-restrictions that a reasonable (and even novice) user could successfully insert the encode and decode SPARK 2014 procedures and functions into their CubedOS project with minor to no diﬃculty. XDR2OS3 takes as input an MXDR ﬁle which currently supports the types: Unsigned Integer, Integer, Float, Enumeration, Opaque Data, String, Structure, and Typedef variations of these types as deﬁned in RFC-4506 [3] (it will accept other types as well as outlined in Appendix A, but will not necessarily give a verifiable SPARK solution as of yet—arbitrary XDR types may be used in the future to signify somewhat arbitrary but sensible output as needed).

**6.2 Design**

**6.2.1 System Level Design**

****

**6.2.2 Component Interactions**

The user of CubedOS creates an MXDR ﬁle intended to generate two SPARK 2014 output ﬁles: \*.ads and concomitant \*.adb. Internally, the lexical analyzer and parser (with the generated parse tree) have all been automatically generated in Scala based on the supplied grammar. The MXDR is first passed by a file that populates several Scala maps and outputs an error message if any IDENTIFIER is a lower-case match of any reserved Ada word. Next, additional semantic analysis is performed. After that, SPARK 2014 (\*.ads and \*.adb) is generated based on the correctness of the original MXDR input, the parse tree and Scala symbol table maps heavily referenced in the process.

**6.2.3 How It Works**

The user of CubedOS speciﬁes through the MXDR language which modules will be able to encode and/or decode which kinds of CubedOS messages. XDR2OS3 will then generate provable SPARK 2014 for each module, based on the module and its capabilities. Using this document as reference, the user of CubedOS speciﬁes for a particular module what it can handle in terms of encoding and decoding a subset of the pre-deﬁned kinds of CubedOS messages. The tool, XDR2OS3, is then invoked over the appropriate directory using a command-line (CMD) syntax as illustrated in Appendix B. XDR2OS3 then outputs provable SPARK 2014 which can be manually placed into a project. The code is then statically tested. See section 6.2.5 for four complete examples of an MXDR file and its two output files.

**6.2.4 Types**

1. **Structured:**

* Enum:

enum {red, green, blue, …} Color; // Identifier may be placed

enum {red, green, blue, …} Color[ARR\_SIZE]; // on either side of curly braces,

enum Series\_Type { One\_Shot, Periodoc }; //but only to the right if an array!

* Struct:
* **Structured subtypes:**
* Struct:

struct bat {

int h;

float j;

};

struct cat {

bat u;

int i;

int y;

};

struct {

int i[100];

int y;

} sat;

struct {

int i[100];

int y;

} sat[ARR\_SIZE];

\*When declaring arrays of structured types, the identifier comes AFTER the closing curly-brace.

* Enum:

See Structured type example for enum above.

* **Unstructured subtypes:**
* Int and unsigned int, double and float:

See Structured subtypes example above. Structs may be nested in any order. Ints, unsigned ints and floats may be previously defined either as a regular type or a typedef or inserted on-the-fly into the struct as a regular type.

* Hyper and unsigned hyper, string, boolean, opaque data, time types:

See Unstructured types below. Each may be either declared “on-the-fly” or by using a previously defined type.

* Message struct:

message struct -> Example {

Width\_Type thing;

sat cube;

};

message struct <- Example2 {

void;

};

‘void;’ may be used as the only message struct parameter to form a basic message-handling skeleton.

1. **Unstructured:**

* Int:

typedef int Hello;

typedef int Hello = “value”;

typedef int Hello range “value” .. “value”;

int Hello;

int Hello = “value”;

typedef int Hello[ARR\_SIZE];

int Hello[ARR\_SIZE];

* Unsigned int:

typedef unsigned int Hello;

typedef unsigned int Hello = “value”;

typedef unsigned int Hello range “value” .. “value”;

unsigned int Hello;

unsigned int Hello = “value”;

typedef unsigned int Hello[ARR\_SIZE];

unsigned int Hello[ARR\_SIZE];

* Float:

typedef float Hello;

float Hello;

typedef float Hello[ARR\_SIZE];

float Hello[ARR\_SIZE];

* Double:

typedef double Hello;

double Hello;

typedef double Hello[ARR\_SIZE];

double Hello[ARR\_SIZE];

* Hyper:

typedef hyper Hello;

typedef hyper Hello = “value”;

typedef hyper Hello range “value” .. “value”;

hyper Hello;

hyper Hello = “value”;

typedef hyper Hello[ARR\_SIZE];

hyper Hello[ARR\_SIZE];

* Unsigned hyper:

typedef unsigned hyper Hello;

typedef unsigned hyper Hello = “value”;

typedef unsigned hyper Hello range “value” .. “value”;

unsigned hyper Hello;

unsigned hyper Hello = “value”;

typedef unsigned hyper Hello[ARR\_SIZE];

unsigned hyper Hello[ARR\_SIZE];

* Opaque:

opaque Message\_Data;

typedef opaque Message\_Data;

* Boolean:

bool b;

typedef bool b;

typedef bool b[ARR\_SIZE];

bool b[ARR\_SIZE];

* String:

string str;

typedef string str;

typedef string str[ARR\_SIZE];

string str[ARR\_SIZE];

* Constant:

const id;

const id = “value”;

* Built-in Time and Time\_Span types:

enum Series\_Type { One\_Shot, Periodoc };

//Natural Ada designation built-in.

//Ada ‘Last syntax may be used for convenience to specify upper bounds

//for both the Natural designation and previously defined variables.

typedef unsigned int Series\_ID\_Type range 1 .. Natural'Last;

typedef unsigned int Series\_Count\_Type range 0 .. Natural'Last;

message struct -> Relative\_Request{

Ada.Real\_Time.Time\_Span Time\_Span;

Series\_Type Request\_Type;

Series\_ID\_Type Series\_ID;

};

message struct -> Absolute\_Request{

Ada.Real\_Time.Time Time;

Series\_ID\_Type Series\_ID;

};

**6.2.5 How To Use**

* **MXDR file**

1. The currently available typedefs or declared types are ‘unsigned int’, ‘int’, ‘float’, ‘double’, ‘hyper’, ‘unsigned hyper’, ‘opaque’, ‘enum’, ‘struct’ and ‘string’—all available as an array of specified size. Constant declarations are also available. Typedefs may be of another type and be associated with a range. Ranges can be specified using another typedef’s maximum range size or by specifying ‘Natural’—both with the SPARK-like syntax ‘‘Last’. Opaque declarations are reserved for SPARK array types used to signify the concept of Data: typedef opaque *IDENTIFIER*<>, with an optional value between the angles. Optionally, all the same functionality is available without the XDR ‘typedef’ specification.
2. Message structs are specified as to whether their generated subprograms are transferring or receiving data with either a ‘->’ (receiving) or ‘<-‘ (sending) symbol. Message invariants may be appended to the end of the message struct after the closing curly brace.
3. The following is MXDR1 .mxdr file for a fictitious module API:

enum {red, green, blue} Color;

typedef unsigned int Hello range 1 .. 2;

const gorilla = 2;

const whatever = 1;

typedef double arnold[10];

typedef bool boo;

typedef hyper hyp range 1 .. 200;

typedef unsigned hyper hyp2 range 1 .. 200;

typedef unsigned int Width\_Type range 1 .. 100;

typedef unsigned int horse is Width\_Type range 1 .. 10;

typedef unsigned int banana range 0 .. whatever;

typedef unsigned int grape range 1 .. gorilla;

typedef unsigned int seed range 0 .. grape'Last;

typedef unsigned int stem is seed range 0 .. seed'Last;

typedef stem branch[10000];

struct bat {

Color h;

Width\_Type j;

branch Tree\_Limb;

};

struct cat {

bat u;

double i;

unsigned int y;

};

struct sat {

int k;

cat led;

bat u;

};

typedef enum {yellow, orange, violet} Other\_Colors;

message struct -> Reddish {

Color i;

int y;

banana b;

boo bo;

hyp2 hypey2;

};

message struct <- Bluish {

Other\_Colors r;

grape g;

arnold a;

float schwarz;

seed s;

};

message struct -> Example {

Width\_Type thing;

sat cube;

hyp hypey;

};

1. The desired specification output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-MXDR1-api.ads**

**-- SUBJECT: Specification of a package that defines the MXDR1 API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with CubedOS.Lib;**

**with Message\_Manager; use Message\_Manager;**

**with System;**

**package CubedOS.MXDR1.API is**

**type Message\_Type is**

**(Example,**

**Bluish,**

**Reddish);**

**type Color is**

**(red,**

**green,**

**blue);**

**type Hello is new Lib.Quadruple\_Octet range 1 .. 2;**

**gorilla: constant := 2;**

**whatever: constant := 1;**

**type arnold is array (0 .. 10) of Double;**

**type boo is new Boolean;**

**type hyp is new Lib.Hyper\_Type range 1 .. 200;**

**type hyp2 is new Lib.U\_Hyper\_Type range 1 .. 200;**

**type Width\_Type is new Lib.Quadruple\_Octet range 1 .. 100;**

**subtype horse is Width\_Type range 1 .. 10;**

**type banana is new Lib.Quadruple\_Octet range 0 .. 1;**

**type grape is new Lib.Quadruple\_Octet range 1 .. 2;**

**type seed is new Lib.Quadruple\_Octet range 0 .. 2.0;**

**subtype stem is seed range 0 .. 2.0;**

**type branch is array (0 .. 10000) of stem;**

**type bat is**

**record**

**h : Color;**

**j : Width\_Type;**

**Tree\_Limb : branch;**

**end record;**

**type cat is**

**record**

**u : bat;**

**i : Double;**

**y : Lib.Quadruple\_Octet;**

**end record;**

**type sat is**

**record**

**k : Integer;**

**led : cat;**

**u : bat;**

**end record;**

**type Other\_Colors is**

**(yellow,**

**orange,**

**violet);**

**function Reddish\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**i : Color;**

**y : Integer;**

**b : banana;**

**bo : boo;**

**hypey2 : hyp2;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Reddish(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Reddish));**

**procedure Reddish\_Decode**

**(Message : in Message\_Record;**

**i : out Color;**

**y : out Integer;**

**b : out banana;**

**bo : out boo;**

**hypey2 : out hyp2;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Reddish(Message),**

**Depends => ((i, y, b, bo, hypey2, Decode\_Status) => Message);**

**function Bluish\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**r : Other\_Colors;**

**g : grape;**

**a : arnold;**

**schwarz : Float;**

**s : seed;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Bluish(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Bluish));**

**procedure Bluish\_Decode**

**(Message : in Message\_Record;**

**r : out Other\_Colors;**

**g : out grape;**

**a : out arnold;**

**schwarz : out Float;**

**s : out seed;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Bluish(Message),**

**Depends => ((r, g, a, schwarz, s, Decode\_Status) => Message);**

**function Example\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**thing : Width\_Type;**

**cube : sat;**

**hypey : hyp;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Example(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Example));**

**procedure Example\_Decode**

**(Message : in Message\_Record;**

**thing : out Width\_Type;**

**cube : out sat;**

**hypey : out hyp;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Example(Message),**

**Depends => ((thing, cube, hypey, Decode\_Status) => Message);**

**end CubedOS.MXDR1.API;**

1. The desired body output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-MXDR1-api.adb**

**-- SUBJECT: Body of a package that implements the MXDR1 API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with CubedOS.Lib.XDR;**

**with CubedOS.Lib;**

**use CubedOS.Lib;**

**use CubedOS.Lib.XDR;**

**package body CubedOS.MXDR1.API is**

**function Reddish\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**i : Color;**

**y : Integer;**

**b : banana;**

**bo : boo;**

**hypey2 : hyp2;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Reddish),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Color'Pos(i)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Integer(y), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(b), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Boolean'Val(boo'Pos(bo)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned\_Hyper(hypey2), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Reddish\_Encode;**

**procedure Reddish\_Decode**

**(Message : in Message\_Record;**

**i : out Color;**

**y : out Integer;**

**b : out banana;**

**bo : out boo;**

**hypey2 : out hyp2;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_i : XDR.XDR\_Unsigned;**

**Raw\_y : XDR.XDR\_Integer;**

**Raw\_b : XDR.XDR\_Unsigned;**

**Raw\_bo : XDR.XDR\_Boolean;**

**Raw\_hypey2 : XDR.XDR\_Unsigned\_Hyper;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**y := Integer(XDR.XDR\_Integer'First);**

**b := banana'First;**

**bo := boo'First;**

**hypey2 := hyp2'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_i, Last);**

**Position := Last + 1;**

**if Raw\_i in Color'Pos(Color'First) .. Color'Pos(Color'Last) then**

**i := Color'Val(Raw\_i);**

**else**

**Decode\_Status := Malformed;**

**i := Color'First;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_y, Last);**

**Position := Last + 1;**

**if Raw\_y in XDR.XDR\_Integer(Integer'First) .. XDR.XDR\_Integer(Integer'Last) then**

**y := Integer(Raw\_y);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_b, Last);**

**Position := Last + 1;**

**if Raw\_b in XDR.XDR\_Unsigned(banana'First) .. XDR.XDR\_Unsigned(banana'Last) then**

**b := banana(Raw\_b);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_bo, Last);**

**Position := Last + 1;**

**bo := boo'Val(XDR.XDR\_Boolean'Pos(Raw\_bo));**

**Decode\_Status := Success;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_hypey2, Last);**

**if Raw\_hypey2 in XDR.XDR\_Unsigned\_Hyper(hyp2'First) .. XDR.XDR\_Unsigned\_Hyper(hyp2'Last) then**

**hypey2 := hyp2(Raw\_hypey2);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Reddish\_Decode;**

**function Bluish\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**r : Other\_Colors;**

**g : grape;**

**a : arnold;**

**schwarz : Float;**

**s : seed;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Bluish),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Other\_Colors'Pos(r)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(g), Message.Payload, Position, Last);**

**Position := Last + 1;**

**for I in Integer range 0 .. 10 loop**

**XDR.Encode(XDR.XDR\_Double(a(I), Message.Payload, Position, Last);**

**Position := Last + 1;**

**end loop;**

**XDR.Encode(XDR.XDR\_Float(schwarz), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(s), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Bluish\_Encode;**

**procedure Bluish\_Decode**

**(Message : in Message\_Record;**

**r : out Other\_Colors;**

**g : out grape;**

**a : out arnold;**

**schwarz : out Float;**

**s : out seed;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_r : XDR.XDR\_Unsigned;**

**Raw\_g : XDR.XDR\_Unsigned;**

**Raw\_a : XDR.XDR\_Double;**

**Raw\_schwarz : XDR.XDR\_Float;**

**Raw\_s : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**g := grape'First;**

**for I in Integer range 0 .. 10 loop**

**a(I) := arnold'First;**

**end loop;**

**schwarz := Float'First;**

**s := seed'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_r, Last);**

**Position := Last + 1;**

**if Raw\_r in Other\_Colors'Pos(Other\_Colors'First) .. Other\_Colors'Pos(Other\_Colors'Last) then**

**r := Other\_Colors'Val(Raw\_r);**

**else**

**Decode\_Status := Malformed;**

**r := Other\_Colors'First;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_g, Last);**

**Position := Last + 1;**

**if Raw\_g in XDR.XDR\_Unsigned(grape'First) .. XDR.XDR\_Unsigned(grape'Last) then**

**g := grape(Raw\_g);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**for I in Integer range 0 .. 10 loop**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_a, Last);**

**Position := Last + 1;**

**if Raw\_a in XDR.XDR\_Double(arnold'First) .. XDR.XDR\_Double(arnold'Last) then**

**a(I) := arnold(Raw\_a);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end loop;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_schwarz, Last);**

**Position := Last + 1;**

**if Raw\_schwarz in XDR.XDR\_Float(Float'First) .. XDR.XDR\_Float(Float'Last) then**

**schwarz := Float(Raw\_schwarz);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_s, Last);**

**if Raw\_s in XDR.XDR\_Unsigned(seed'First) .. XDR.XDR\_Unsigned(seed'Last) then**

**s := seed(Raw\_s);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Bluish\_Decode;**

**function Example\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**thing : Width\_Type;**

**cube : sat;**

**hypey : hyp;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Example),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(thing), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Integer(cube.k), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Color'Pos(cube.led.u.h)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(cube.led.u.j), Message.Payload, Position, Last);**

**Position := Last + 1;**

**for I in Integer range 0 .. 10000 loop**

**XDR.Encode(XDR.XDR\_Unsigned(cube.led.u.Tree\_Limb(I)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**end loop;**

**XDR.Encode(XDR.XDR\_Double(cube.led.i), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(cube.led.y), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Color'Pos(cube.u.h)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(cube.u.j), Message.Payload, Position, Last);**

**Position := Last + 1;**

**for I in Integer range 0 .. 10000 loop**

**XDR.Encode(XDR.XDR\_Unsigned(cube.u.Tree\_Limb(I)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**end loop;**

**XDR.Encode(XDR.XDR\_Hyper(hypey), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Example\_Encode;**

**procedure Example\_Decode**

**(Message : in Message\_Record;**

**thing : out Width\_Type;**

**cube : out sat;**

**hypey : out hyp;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_thing : XDR.XDR\_Unsigned;**

**Raw\_cube\_k : XDR.XDR\_Integer;**

**Raw\_cube\_led\_u\_h : XDR.XDR\_Unsigned;**

**Raw\_cube\_led\_u\_j : XDR.XDR\_Unsigned;**

**Raw\_cube\_led\_u\_Tree\_Limb : XDR.XDR\_Unsigned;**

**Raw\_cube\_led\_i : XDR.XDR\_Double;**

**Raw\_cube\_led\_y : XDR.XDR\_Unsigned;**

**Raw\_cube\_u\_h : XDR.XDR\_Unsigned;**

**Raw\_cube\_u\_j : XDR.XDR\_Unsigned;**

**Raw\_cube\_u\_Tree\_Limb : XDR.XDR\_Unsigned;**

**Raw\_hypey : XDR.XDR\_Hyper;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**thing := Width\_Type'First;**

**cube.k := Integer(XDR.XDR\_Integer'First);**

**cube.led.u.h := Color'First;**

**cube.led.u.j := Width\_Type'First;**

**for I in Integer range 0 .. 10000 loop**

**cube.led.u.Tree\_Limb(I) := branch'First;**

**end loop;**

**cube.led.i := Double'First;**

**cube.led.y := Lib.Quadruple\_Octet(XDR.XDR\_Unsigned'First);**

**cube.u.h := Color'First;**

**cube.u.j := Width\_Type'First;**

**for I in Integer range 0 .. 10000 loop**

**cube.u.Tree\_Limb(I) := branch'First;**

**end loop;**

**hypey := hyp'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_thing, Last);**

**Position := Last + 1;**

**if Raw\_thing in XDR.XDR\_Unsigned(Width\_Type'First) .. XDR.XDR\_Unsigned(Width\_Type'Last) then**

**thing := Width\_Type(Raw\_thing);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_k, Last);**

**Position := Last + 1;**

**if Raw\_cube\_k in XDR.XDR\_Integer(Integer'First) .. XDR.XDR\_Integer(Integer'Last) then**

**cube.k := Integer(Raw\_cube\_k);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_led\_u\_h, Last);**

**Position := Last + 1;**

**if Raw\_cube\_led\_u\_h in Color'Pos(Color'First) .. Color'Pos(Color'Last) then**

**cube.led.u.h := Color'Val(Raw\_cube\_led\_u\_h);**

**else**

**Decode\_Status := Malformed;**

**cube.led.u.h := Color'First;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_led\_u\_j, Last);**

**Position := Last + 1;**

**if Raw\_cube\_led\_u\_j in XDR.XDR\_Unsigned(Width\_Type'First) .. XDR.XDR\_Unsigned(Width\_Type'Last) then**

**cube.led.u.j := Width\_Type(Raw\_cube\_led\_u\_j);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**for I in Integer range 0 .. 10000 loop**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_led\_u\_Tree\_Limb, Last);**

**Position := Last + 1;**

**if Raw\_cube\_led\_u\_Tree\_Limb in XDR.XDR\_Unsigned(Lib.Quadruple\_Octet'First) .. XDR.XDR\_Unsigned(Lib.Quadruple\_Octet'Last) then**

**cube.led.u.Tree\_Limb(I) := branch(Raw\_cube\_led\_u\_Tree\_Limb);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end loop;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_led\_i, Last);**

**Position := Last + 1;**

**if Raw\_cube\_led\_i in XDR.XDR\_Double(Double'First) .. XDR.XDR\_Double(Double'Last) then**

**cube.led.i := Double(Raw\_cube\_led\_i);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_led\_y, Last);**

**Position := Last + 1;**

**if Raw\_cube\_led\_y in XDR.XDR\_Unsigned(Lib.Quadruple\_Octet'First) .. XDR.XDR\_Unsigned(Lib.Quadruple\_Octet'Last) then**

**cube.led.y := Lib.Quadruple\_Octet(Raw\_cube\_led\_y);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_u\_h, Last);**

**Position := Last + 1;**

**if Raw\_cube\_u\_h in Color'Pos(Color'First) .. Color'Pos(Color'Last) then**

**cube.u.h := Color'Val(Raw\_cube\_u\_h);**

**else**

**Decode\_Status := Malformed;**

**cube.u.h := Color'First;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_u\_j, Last);**

**Position := Last + 1;**

**if Raw\_cube\_u\_j in XDR.XDR\_Unsigned(Width\_Type'First) .. XDR.XDR\_Unsigned(Width\_Type'Last) then**

**cube.u.j := Width\_Type(Raw\_cube\_u\_j);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**for I in Integer range 0 .. 10000 loop**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_cube\_u\_Tree\_Limb, Last);**

**Position := Last + 1;**

**if Raw\_cube\_u\_Tree\_Limb in XDR.XDR\_Unsigned(Lib.Quadruple\_Octet'First) .. XDR.XDR\_Unsigned(Lib.Quadruple\_Octet'Last) then**

**cube.u.Tree\_Limb(I) := branch(Raw\_cube\_u\_Tree\_Limb);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end loop;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_hypey, Last);**

**if Raw\_hypey in XDR.XDR\_Hyper(hyp'First) .. XDR.XDR\_Hyper(hyp'Last) then**

**hypey := hyp(Raw\_hypey);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Example\_Decode;**

**end CubedOS.MXDR1.API;**

1. The following is File\_Server .mxdr file for a fictitious module API:

//File\_Server

enum Mode\_Type { Read, Write };

typedef unsigned int File\_Handle\_Type range 0 .. 64;

typedef unsigned int Valid\_File\_Handle\_Type is File\_Handle\_Type range 1 .. File\_Handle\_Type'Last;

const Invalid\_Handle is File\_Handle\_Type = 0;

const Maximum\_Read\_Size = 256;

const Maximum\_Write\_Size = 256;

typedef unsigned int Read\_Result\_Size\_Type is Natural range 0 .. Maximum\_Read\_Size;

typedef unsigned int Read\_Size\_Type is Natural range 1 .. Read\_Result\_Size\_Type'Last;

typedef unsigned int Write\_Result\_Size\_Type is Natural range 0 .. Maximum\_Write\_Size;

typedef unsigned int Write\_Size\_Type is Natural range 1 .. Write\_Result\_Size\_Type'Last;

typedef unsigned int Request\_ID\_Type range 1 .. 64;

message struct -> Open\_Request {

Mode\_Type Mode;

string Name;

Request\_ID\_Type Request\_ID;

};

message struct <- Open\_Reply {

File\_Handle\_Type Handle;

Request\_ID\_Type Request\_ID;

};

message struct -> Read\_Request{

File\_Handle\_Type Handle;

Read\_Size\_Type Amount;

};

message struct <- Read\_Reply {

Valid\_File\_Handle\_Type Handle;

Read\_Result\_Size\_Type Amount;

CubedOS.Lib.Octet\_Array Message\_Data<>;

} with message\_invariant => Amount <= Message\_Data'Length;

message struct -> Write\_Request {

Valid\_File\_Handle\_Type Handle;

Write\_Size\_Type Amount;

CubedOS.Lib.Octet\_Array Message\_Data<>;

} with message\_invariant => Amount <= Message\_Data'Length;

message struct <- Write\_Reply {

Valid\_File\_Handle\_Type Handle;

Write\_Result\_Size\_Type Amount;

};

message struct -> Close\_Request {

Valid\_File\_Handle\_Type Handle;

};

1. The desired specification output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-File\_Server-api.ads**

**-- SUBJECT: Specification of a package that defines the File\_Server API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with CubedOS.Lib;**

**with Message\_Manager; use Message\_Manager;**

**with System;**

**package CubedOS.File\_Server.API is**

**type Message\_Type is**

**(Read\_Reply,**

**Open\_Request,**

**Close\_Request,**

**Read\_Request,**

**Write\_Reply,**

**Open\_Reply,**

**Write\_Request);**

**type Mode\_Type is**

**(Read,**

**Write);**

**type File\_Handle\_Type is new Lib.Quadruple\_Octet range 0 .. 64;**

**subtype Valid\_File\_Handle\_Type is File\_Handle\_Type range 1 .. 64.0;**

**Invalid\_Handle: constant File\_Handle\_Type := 0;**

**Maximum\_Read\_Size: constant := 256;**

**Maximum\_Write\_Size: constant := 256;**

**subtype Read\_Result\_Size\_Type is Natural range 0 .. 256;**

**subtype Read\_Size\_Type is Read\_Result\_Size\_Type range 1 .. 256.0;**

**subtype Write\_Result\_Size\_Type is Natural range 0 .. 256;**

**subtype Write\_Size\_Type is Write\_Result\_Size\_Type range 1 .. 256.0;**

**type Request\_ID\_Type is new Lib.Quadruple\_Octet range 1 .. 64;**

**function Open\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Mode : Mode\_Type;**

**Name : String;**

**Request\_ID : Request\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null,**

**Pre => (0 < Name'Length and Name'Length <= XDR\_Size\_Type'Last - 12);**

**function Is\_Open\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Open\_Request));**

**procedure Open\_Request\_Decode**

**(Message : in Message\_Record;**

**Mode : out Mode\_Type;**

**Name : out String;**

**Name\_Size : out Natural;**

**Request\_ID : out Request\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Open\_Request(Message),**

**Depends => ((Mode, Name, Name\_Size, Request\_ID, Decode\_Status) => Message);**

**function Open\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Handle : File\_Handle\_Type;**

**Request\_ID : Request\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Open\_Reply(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Open\_Reply));**

**procedure Open\_Reply\_Decode**

**(Message : in Message\_Record;**

**Handle : out File\_Handle\_Type;**

**Request\_ID : out Request\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Open\_Reply(Message),**

**Depends => ((Handle, Request\_ID, Decode\_Status) => Message);**

**function Read\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Handle : File\_Handle\_Type;**

**Amount : Read\_Size\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Read\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Read\_Request));**

**procedure Read\_Request\_Decode**

**(Message : in Message\_Record;**

**Handle : out File\_Handle\_Type;**

**Amount : out Read\_Size\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Read\_Request(Message),**

**Depends => ((Handle, Amount, Decode\_Status) => Message);**

**function Read\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Amount : Read\_Result\_Size\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null,**

**Pre => Amount <= Message\_Data'Length;**

**function Is\_Read\_Reply(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Read\_Reply));**

**procedure Read\_Reply\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Amount : out Read\_Result\_Size\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Read\_Reply(Message),**

**Depends => ((Handle, Amount, Message\_Data, Size, Decode\_Status) => Message),**

**Post => Amount <= Message\_Data'Length;**

**function Write\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Amount : Write\_Size\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null,**

**Pre => Amount <= Message\_Data'Length;**

**function Is\_Write\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Write\_Request));**

**procedure Write\_Request\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Amount : out Write\_Size\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Write\_Request(Message),**

**Depends => ((Handle, Amount, Message\_Data, Size, Decode\_Status) => Message),**

**Post => Amount <= Message\_Data'Length;**

**function Write\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Amount : Write\_Result\_Size\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Write\_Reply(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Write\_Reply));**

**procedure Write\_Reply\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Amount : out Write\_Result\_Size\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Write\_Reply(Message),**

**Depends => ((Handle, Amount, Decode\_Status) => Message);**

**function Close\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Close\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Close\_Request));**

**procedure Close\_Request\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Close\_Request(Message),**

**Depends => ((Handle, Decode\_Status) => Message);**

**end CubedOS.File\_Server.API;**

1. The desired body output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-File\_Server-api.adb**

**-- SUBJECT: Body of a package that implements the File\_Server API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with CubedOS.Lib.XDR;**

**with CubedOS.Lib;**

**use CubedOS.Lib;**

**use CubedOS.Lib.XDR;**

**package body CubedOS.File\_Server.API is**

**function Open\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Mode : Mode\_Type;**

**Name : String;**

**Request\_ID : Request\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Open\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Mode\_Type'Pos(Mode)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Name'Length), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(Name, Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Request\_ID), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Open\_Request\_Encode;**

**procedure Open\_Request\_Decode**

**(Message : in Message\_Record;**

**Mode : out Mode\_Type;**

**Name : out String;**

**Name\_Size : out Natural;**

**Request\_ID : out Request\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Mode : XDR.XDR\_Unsigned;**

**Raw\_Name\_Size : XDR.XDR\_Unsigned;**

**Raw\_Request\_ID : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Name := (others => ' ');**

**Request\_ID := Request\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Mode, Last);**

**Position := Last + 1;**

**if Raw\_Mode in Mode\_Type'Pos(Mode\_Type'First) .. Mode\_Type'Pos(Mode\_Type'Last) then**

**Mode := Mode\_Type'Val(Raw\_Mode);**

**else**

**Decode\_Status := Malformed;**

**Mode := Mode\_Type'First;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Name\_Size, Last);**

**Position := Last + 1;**

**if Raw\_Name\_Size in XDR.XDR\_Unsigned(Natural'First) .. XDR.XDR\_Unsigned(Natural'Last) then**

**Name\_Size := Natural(Raw\_Name\_Size);**

**else**

**Name\_Size := 0;**

**end if;**

**if Name\_Size < 1 then**

**XDR.Decode(Message.Payload, Position, Name(Name'First .. Name'First + (Name\_Size - 1)), Last);**

**Position := Last + 1;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Request\_ID, Last);**

**if Raw\_Request\_ID in XDR.XDR\_Unsigned(Request\_ID\_Type'First) .. XDR.XDR\_Unsigned(Request\_ID\_Type'Last) then**

**Request\_ID := Request\_ID\_Type(Raw\_Request\_ID);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Open\_Request\_Decode;**

**function Open\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Handle : File\_Handle\_Type;**

**Request\_ID : Request\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Open\_Reply),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Handle), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Request\_ID), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Open\_Reply\_Encode;**

**procedure Open\_Reply\_Decode**

**(Message : in Message\_Record;**

**Handle : out File\_Handle\_Type;**

**Request\_ID : out Request\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Handle : XDR.XDR\_Unsigned;**

**Raw\_Request\_ID : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Handle := File\_Handle\_Type'First;**

**Request\_ID := Request\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Handle, Last);**

**Position := Last + 1;**

**if Raw\_Handle in XDR.XDR\_Unsigned(File\_Handle\_Type'First) .. XDR.XDR\_Unsigned(File\_Handle\_Type'Last) then**

**Handle := File\_Handle\_Type(Raw\_Handle);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Request\_ID, Last);**

**if Raw\_Request\_ID in XDR.XDR\_Unsigned(Request\_ID\_Type'First) .. XDR.XDR\_Unsigned(Request\_ID\_Type'Last) then**

**Request\_ID := Request\_ID\_Type(Raw\_Request\_ID);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Open\_Reply\_Decode;**

**function Read\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Handle : File\_Handle\_Type;**

**Amount : Read\_Size\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Read\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Handle), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Amount), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Read\_Request\_Encode;**

**procedure Read\_Request\_Decode**

**(Message : in Message\_Record;**

**Handle : out File\_Handle\_Type;**

**Amount : out Read\_Size\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Handle : XDR.XDR\_Unsigned;**

**Raw\_Amount : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Handle := File\_Handle\_Type'First;**

**Amount := Read\_Size\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Handle, Last);**

**Position := Last + 1;**

**if Raw\_Handle in XDR.XDR\_Unsigned(File\_Handle\_Type'First) .. XDR.XDR\_Unsigned(File\_Handle\_Type'Last) then**

**Handle := File\_Handle\_Type(Raw\_Handle);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Amount, Last);**

**if Raw\_Amount in XDR.XDR\_Unsigned(Read\_Size\_Type'First) .. XDR.XDR\_Unsigned(Read\_Size\_Type'Last) then**

**Amount := Read\_Size\_Type(Raw\_Amount);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Read\_Request\_Decode;**

**function Read\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Amount : Read\_Result\_Size\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Read\_Reply),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Handle), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Amount), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Message\_Data'Length), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(Message\_Data, Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Read\_Reply\_Encode;**

**procedure Read\_Reply\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Amount : out Read\_Result\_Size\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Handle : XDR.XDR\_Unsigned;**

**Raw\_Amount : XDR.XDR\_Unsigned;**

**Raw\_Size : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Handle := Valid\_File\_Handle\_Type'First;**

**Amount := Read\_Result\_Size\_Type'First;**

**Message\_Data := (others => 0);**

**Size := 0;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Handle, Last);**

**Position := Last + 1;**

**if Raw\_Handle in XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'First) .. XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'Last) then**

**Handle := Valid\_File\_Handle\_Type(Raw\_Handle);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Amount, Last);**

**Position := Last + 1;**

**if Raw\_Amount in XDR.XDR\_Unsigned(Read\_Result\_Size\_Type'First) .. XDR.XDR\_Unsigned(Read\_Result\_Size\_Type'Last) then**

**Amount := Read\_Result\_Size\_Type(Raw\_Amount);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Size, Last);**

**Position := Last + 1;**

**if Raw\_Size in XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'First) .. XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'Last) then**

**Size := CubedOS.Lib.Octet\_Array\_Count(Raw\_Size);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**if Size < Message\_Data'Length then**

**XDR.Decode(Message.Payload, Position, Message\_Data(Message\_Data'First .. Message\_Data'First + Size - 1), Last);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Read\_Reply\_Decode;**

**function Write\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Amount : Write\_Size\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Write\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Handle), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Amount), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Message\_Data'Length), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(Message\_Data, Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Write\_Request\_Encode;**

**procedure Write\_Request\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Amount : out Write\_Size\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Handle : XDR.XDR\_Unsigned;**

**Raw\_Amount : XDR.XDR\_Unsigned;**

**Raw\_Size : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Handle := Valid\_File\_Handle\_Type'First;**

**Amount := Write\_Size\_Type'First;**

**Message\_Data := (others => 0);**

**Size := 0;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Handle, Last);**

**Position := Last + 1;**

**if Raw\_Handle in XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'First) .. XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'Last) then**

**Handle := Valid\_File\_Handle\_Type(Raw\_Handle);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Amount, Last);**

**Position := Last + 1;**

**if Raw\_Amount in XDR.XDR\_Unsigned(Write\_Size\_Type'First) .. XDR.XDR\_Unsigned(Write\_Size\_Type'Last) then**

**Amount := Write\_Size\_Type(Raw\_Amount);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Size, Last);**

**Position := Last + 1;**

**if Raw\_Size in XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'First) .. XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'Last) then**

**Size := CubedOS.Lib.Octet\_Array\_Count(Raw\_Size);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**if Size < Message\_Data'Length then**

**XDR.Decode(Message.Payload, Position, Message\_Data(Message\_Data'First .. Message\_Data'First + Size - 1), Last);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Write\_Request\_Decode;**

**function Write\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Amount : Write\_Result\_Size\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Write\_Reply),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Handle), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Amount), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Write\_Reply\_Encode;**

**procedure Write\_Reply\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Amount : out Write\_Result\_Size\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Handle : XDR.XDR\_Unsigned;**

**Raw\_Amount : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Handle := Valid\_File\_Handle\_Type'First;**

**Amount := Write\_Result\_Size\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Handle, Last);**

**Position := Last + 1;**

**if Raw\_Handle in XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'First) .. XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'Last) then**

**Handle := Valid\_File\_Handle\_Type(Raw\_Handle);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Amount, Last);**

**if Raw\_Amount in XDR.XDR\_Unsigned(Write\_Result\_Size\_Type'First) .. XDR.XDR\_Unsigned(Write\_Result\_Size\_Type'Last) then**

**Amount := Write\_Result\_Size\_Type(Raw\_Amount);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Write\_Reply\_Decode;**

**function Close\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Handle : Valid\_File\_Handle\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Close\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Handle), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Close\_Request\_Encode;**

**procedure Close\_Request\_Decode**

**(Message : in Message\_Record;**

**Handle : out Valid\_File\_Handle\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Handle : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Handle := Valid\_File\_Handle\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Handle, Last);**

**if Raw\_Handle in XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'First) .. XDR.XDR\_Unsigned(Valid\_File\_Handle\_Type'Last) then**

**Handle := Valid\_File\_Handle\_Type(Raw\_Handle);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Close\_Request\_Decode;**

**end CubedOS.File\_Server.API;**

1. The following is Tick\_Generator .mxdr file for a fictitious module API:

//tick generator

enum Series\_Type { One\_Shot, Periodoc };

typedef unsigned int Series\_ID\_Type range 1 .. Natural'Last;

typedef unsigned int Series\_Count\_Type range 0 .. Natural'Last;

message struct -> Relative\_Request{

Ada.Real\_Time.Time\_Span Time\_Span;

Series\_Type Request\_Type;

Series\_ID\_Type Series\_ID;

};

message struct -> Absolute\_Request{

Ada.Real\_Time.Time Time;

Series\_ID\_Type Series\_ID;

};

message struct <- Tick\_Reply{

Series\_ID\_Type Series\_ID;

Series\_Count\_Type Count;

};

message struct -> Cancel\_Request{

Series\_ID\_Type Series\_ID;

};

1. The desired specification output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-Tick\_Generator-api.ads**

**-- SUBJECT: Specification of a package that defines the Tick\_Generator API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with Ada.Real\_Time;**

**with CubedOS.Lib;**

**with Message\_Manager; use Message\_Manager;**

**with System;**

**package CubedOS.Tick\_Generator.API is**

**type Message\_Type is**

**(Relative\_Request,**

**Absolute\_Request,**

**Cancel\_Request,**

**Tick\_Reply);**

**type Series\_Type is**

**(One\_Shot,**

**Periodoc);**

**type Series\_ID\_Type is new Lib.Quadruple\_Octet;**

**type Series\_Count\_Type is new Lib.Quadruple\_Octet;**

**function Relative\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Time\_Span : Ada.Real\_Time.Time\_Span;**

**Request\_Type : Series\_Type;**

**Series\_ID : Series\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Relative\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Relative\_Request));**

**procedure Relative\_Request\_Decode**

**(Message : in Message\_Record;**

**Time\_Span : out Ada.Real\_Time.Time\_Span;**

**Request\_Type : out Series\_Type;**

**Series\_ID : out Series\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Relative\_Request(Message),**

**Depends => ((Time\_Span, Request\_Type, Series\_ID, Decode\_Status) => Message);**

**function Absolute\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Time : Ada.Real\_Time.Time;**

**Series\_ID : Series\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Absolute\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Absolute\_Request));**

**procedure Absolute\_Request\_Decode**

**(Message : in Message\_Record;**

**Time : out Ada.Real\_Time.Time;**

**Series\_ID : out Series\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Absolute\_Request(Message),**

**Depends => ((Time, Series\_ID, Decode\_Status) => Message);**

**function Tick\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Series\_ID : Series\_ID\_Type;**

**Count : Series\_Count\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Tick\_Reply(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Tick\_Reply));**

**procedure Tick\_Reply\_Decode**

**(Message : in Message\_Record;**

**Series\_ID : out Series\_ID\_Type;**

**Count : out Series\_Count\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Tick\_Reply(Message),**

**Depends => ((Series\_ID, Count, Decode\_Status) => Message);**

**function Cancel\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Series\_ID : Series\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Cancel\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Cancel\_Request));**

**procedure Cancel\_Request\_Decode**

**(Message : in Message\_Record;**

**Series\_ID : out Series\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Cancel\_Request(Message),**

**Depends => ((Series\_ID, Decode\_Status) => Message);**

**end CubedOS.Tick\_Generator.API;**

1. The desired body output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-Tick\_Generator-api.adb**

**-- SUBJECT: Body of a package that implements the Tick\_Generator API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with Ada.Real\_Time;**

**use Ada.Real\_Time;**

**with CubedOS.Lib.XDR;**

**with CubedOS.Lib;**

**use CubedOS.Lib;**

**use CubedOS.Lib.XDR;**

**package body CubedOS.Tick\_Generator.API is**

**function Relative\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Time\_Span : Ada.Real\_Time.Time\_Span;**

**Request\_Type : Series\_Type;**

**Series\_ID : Series\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**Interval : constant Duration := Ada.Real\_Time.To\_Duration(Time\_Span);**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Relative\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(1000\*Interval), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Series\_Type'Pos(Request\_Type)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Series\_ID), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Relative\_Request\_Encode;**

**procedure Relative\_Request\_Decode**

**(Message : in Message\_Record;**

**Time\_Span : out Ada.Real\_Time.Time\_Span;**

**Request\_Type : out Series\_Type;**

**Series\_ID : out Series\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Interval : XDR.XDR\_Unsigned;**

**Raw\_Request\_Type : XDR.XDR\_Unsigned;**

**Raw\_Series\_ID : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Time\_Span := Ada.Real\_Time.Time\_Span\_First;**

**Series\_ID := Series\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Interval, Last);**

**Position := Last + 1;**

**if Raw\_Interval < XDR.XDR\_Unsigned(Integer'Last) then**

**Time\_Span := Ada.Real\_Time.Milliseconds(Integer(Raw\_Interval));**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Request\_Type, Last);**

**Position := Last + 1;**

**if Raw\_Request\_Type in Series\_Type'Pos(Series\_Type'First) .. Series\_Type'Pos(Series\_Type'Last) then**

**Request\_Type := Series\_Type'Val(Raw\_Request\_Type);**

**else**

**Decode\_Status := Malformed;**

**Request\_Type := Series\_Type'First;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Series\_ID, Last);**

**if Raw\_Series\_ID in XDR.XDR\_Unsigned(Series\_ID\_Type'First) .. XDR.XDR\_Unsigned(Series\_ID\_Type'Last) then**

**Series\_ID := Series\_ID\_Type(Raw\_Series\_ID);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Relative\_Request\_Decode;**

**function Absolute\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Time : Ada.Real\_Time.Time;**

**Series\_ID : Series\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**Seconds : Ada.Real\_Time.Seconds\_Count;**

**Fraction : Ada.Real\_Time.Time\_Span;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Absolute\_Request),**

**Priority => Priority);**

**Ada.Real\_Time.Split(Time, Seconds, Fraction);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Seconds), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Series\_ID), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Absolute\_Request\_Encode;**

**procedure Absolute\_Request\_Decode**

**(Message : in Message\_Record;**

**Time : out Ada.Real\_Time.Time;**

**Series\_ID : out Series\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Seconds : Ada.Real\_Time.Seconds\_Count;**

**Raw\_Time : XDR.XDR\_Unsigned;**

**Raw\_Series\_ID : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Time := Ada.Real\_Time.Time\_First;**

**Series\_ID := Series\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Time, Last);**

**Position := Last + 1;**

**if Raw\_Time < XDR.XDR\_Unsigned(Integer'Last) then**

**Seconds := Ada.Real\_Time.Seconds\_Count(Raw\_Time);**

**Time := Ada.Real\_Time.Time\_Of(Seconds, Ada.Real\_Time.Time\_Span\_Zero);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Series\_ID, Last);**

**if Raw\_Series\_ID in XDR.XDR\_Unsigned(Series\_ID\_Type'First) .. XDR.XDR\_Unsigned(Series\_ID\_Type'Last) then**

**Series\_ID := Series\_ID\_Type(Raw\_Series\_ID);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Absolute\_Request\_Decode;**

**function Tick\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Series\_ID : Series\_ID\_Type;**

**Count : Series\_Count\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Tick\_Reply),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Series\_ID), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Count), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Tick\_Reply\_Encode;**

**procedure Tick\_Reply\_Decode**

**(Message : in Message\_Record;**

**Series\_ID : out Series\_ID\_Type;**

**Count : out Series\_Count\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Series\_ID : XDR.XDR\_Unsigned;**

**Raw\_Count : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Series\_ID := Series\_ID\_Type'First;**

**Count := Series\_Count\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Series\_ID, Last);**

**Position := Last + 1;**

**if Raw\_Series\_ID in XDR.XDR\_Unsigned(Series\_ID\_Type'First) .. XDR.XDR\_Unsigned(Series\_ID\_Type'Last) then**

**Series\_ID := Series\_ID\_Type(Raw\_Series\_ID);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Count, Last);**

**if Raw\_Count in XDR.XDR\_Unsigned(Series\_Count\_Type'First) .. XDR.XDR\_Unsigned(Series\_Count\_Type'Last) then**

**Count := Series\_Count\_Type(Raw\_Count);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Tick\_Reply\_Decode;**

**function Cancel\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Series\_ID : Series\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Cancel\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Series\_ID), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Cancel\_Request\_Encode;**

**procedure Cancel\_Request\_Decode**

**(Message : in Message\_Record;**

**Series\_ID : out Series\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Series\_ID : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Series\_ID := Series\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Series\_ID, Last);**

**if Raw\_Series\_ID in XDR.XDR\_Unsigned(Series\_ID\_Type'First) .. XDR.XDR\_Unsigned(Series\_ID\_Type'Last) then**

**Series\_ID := Series\_ID\_Type(Raw\_Series\_ID);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Cancel\_Request\_Decode;**

**end CubedOS.Tick\_Generator.API;**

1. The following is Publish\_Subscribe .mxdr file for a fictitious module API:

enum Status\_Type { Success, Failure};

typedef unsigned int Channel\_ID\_Type range 0 .. Natural'Last;

message struct -> Subscribe\_Request {

Channel\_ID\_Type Channel;

};

message struct <- Subscribe\_Reply {

Channel\_ID\_Type Channel;

Status\_Type Status;

};

message struct -> Publish\_Request {

Channel\_ID\_Type Channel;

CubedOS.Lib.Octet\_Array Message\_Data;

};

message struct <- Publish\_Reply {

Channel\_ID\_Type Channel;

Status\_Type Status;

};

message struct -> Publish\_Result {

Channel\_ID\_Type Channel;

CubedOS.Lib.Octet\_Array Message\_Data;

};

1. The desired specification output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-Publish\_Subscribe-api.ads**

**-- SUBJECT: Specification of a package that defines the Publish\_Subscribe API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with CubedOS.Lib;**

**with Message\_Manager; use Message\_Manager;**

**with System;**

**package CubedOS.Publish\_Subscribe.API is**

**type Message\_Type is**

**(Publish\_Request,**

**Publish\_Result,**

**Publish\_Reply,**

**Subscribe\_Request,**

**Subscribe\_Reply);**

**type Status\_Type is**

**(Success,**

**Failure);**

**type Channel\_ID\_Type is new Lib.Quadruple\_Octet;**

**function Subscribe\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Subscribe\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Subscribe\_Request));**

**procedure Subscribe\_Request\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Subscribe\_Request(Message),**

**Depends => ((Channel, Decode\_Status) => Message);**

**function Subscribe\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Status : Status\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Subscribe\_Reply(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Subscribe\_Reply));**

**procedure Subscribe\_Reply\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Status : out Status\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Subscribe\_Reply(Message),**

**Depends => ((Channel, Status, Decode\_Status) => Message);**

**function Publish\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Publish\_Request(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Publish\_Request));**

**procedure Publish\_Request\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Publish\_Request(Message),**

**Depends => ((Channel, Message\_Data, Size, Decode\_Status) => Message);**

**function Publish\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Status : Status\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Publish\_Reply(Message : Message\_Record) return Boolean is**

**(Message.Receiver = ID and Message.Message\_ID = Message\_Type'Pos(Publish\_Reply));**

**procedure Publish\_Reply\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Status : out Status\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Publish\_Reply(Message),**

**Depends => ((Channel, Status, Decode\_Status) => Message);**

**function Publish\_Result\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**with**

**Global => null;**

**function Is\_Publish\_Result(Message : Message\_Record) return Boolean is**

**(Message.Sender = ID and Message.Message\_ID = Message\_Type'Pos(Publish\_Result));**

**procedure Publish\_Result\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**with**

**Global => null,**

**Pre => Is\_Publish\_Result(Message),**

**Depends => ((Channel, Message\_Data, Size, Decode\_Status) => Message);**

**end CubedOS.Publish\_Subscribe.API;**

1. The desired body output:

**--------------------------------------------------------------------------------**

**-- FILE : cubedos-Publish\_Subscribe-api.adb**

**-- SUBJECT: Body of a package that implements the Publish\_Subscribe API**

**-- AUTHOR : (C) Copyright 2017 by Vermont Technical College**

**-- All the subprograms in this package must be task safe. They can be simultaneously**

**--called from multiple tasks. If possible, make every subprogram here a pure function.**

**--------------------------------------------------------------------------------**

**pragma SPARK\_Mode(On);**

**with CubedOS.Lib.XDR;**

**with CubedOS.Lib;**

**use CubedOS.Lib;**

**use CubedOS.Lib.XDR;**

**package body CubedOS.Publish\_Subscribe.API is**

**function Subscribe\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Subscribe\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Channel), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Subscribe\_Request\_Encode;**

**procedure Subscribe\_Request\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Channel : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Channel := Channel\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Channel, Last);**

**if Raw\_Channel in XDR.XDR\_Unsigned(Channel\_ID\_Type'First) .. XDR.XDR\_Unsigned(Channel\_ID\_Type'Last) then**

**Channel := Channel\_ID\_Type(Raw\_Channel);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Subscribe\_Request\_Decode;**

**function Subscribe\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Status : Status\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Subscribe\_Reply),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Channel), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Status\_Type'Pos(Status)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Subscribe\_Reply\_Encode;**

**procedure Subscribe\_Reply\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Status : out Status\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Channel : XDR.XDR\_Unsigned;**

**Raw\_Status : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Channel := Channel\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Channel, Last);**

**Position := Last + 1;**

**if Raw\_Channel in XDR.XDR\_Unsigned(Channel\_ID\_Type'First) .. XDR.XDR\_Unsigned(Channel\_ID\_Type'Last) then**

**Channel := Channel\_ID\_Type(Raw\_Channel);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Status, Last);**

**if Raw\_Status in Status\_Type'Pos(Status\_Type'First) .. Status\_Type'Pos(Status\_Type'Last) then**

**Status := Status\_Type'Val(Raw\_Status);**

**else**

**Decode\_Status := Malformed;**

**Status := Status\_Type'First;**

**end if;**

**end if;**

**end Subscribe\_Reply\_Decode;**

**function Publish\_Request\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Publish\_Request),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Channel), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Message\_Data'Length), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(Message\_Data, Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Publish\_Request\_Encode;**

**procedure Publish\_Request\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Channel : XDR.XDR\_Unsigned;**

**Raw\_Size : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Channel := Channel\_ID\_Type'First;**

**Message\_Data := (others => 0);**

**Size := 0;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Channel, Last);**

**Position := Last + 1;**

**if Raw\_Channel in XDR.XDR\_Unsigned(Channel\_ID\_Type'First) .. XDR.XDR\_Unsigned(Channel\_ID\_Type'Last) then**

**Channel := Channel\_ID\_Type(Raw\_Channel);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Size, Last);**

**Position := Last + 1;**

**if Raw\_Size in XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'First) .. XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'Last) then**

**Size := CubedOS.Lib.Octet\_Array\_Count(Raw\_Size);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**if Size < Message\_Data'Length then**

**XDR.Decode(Message.Payload, Position, Message\_Data(Message\_Data'First .. Message\_Data'First + Size - 1), Last);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Publish\_Request\_Decode;**

**function Publish\_Reply\_Encode**

**(Receiver\_Domain : Domain\_ID\_Type;**

**Receiver : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Status : Status\_Type;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record := Make\_Empty\_Message(**

**Sender\_Domain => Domain\_ID,**

**Receiver\_Domain => Receiver\_Domain,**

**Sender => ID,**

**Receiver => Receiver,**

**Request\_ID => 0,**

**Message\_ID => Message\_Type'Pos(Publish\_Reply),**

**Priority => Priority);**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Channel), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Status\_Type'Pos(Status)), Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Publish\_Reply\_Encode;**

**procedure Publish\_Reply\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Status : out Status\_Type;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Channel : XDR.XDR\_Unsigned;**

**Raw\_Status : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Channel := Channel\_ID\_Type'First;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Channel, Last);**

**Position := Last + 1;**

**if Raw\_Channel in XDR.XDR\_Unsigned(Channel\_ID\_Type'First) .. XDR.XDR\_Unsigned(Channel\_ID\_Type'Last) then**

**Channel := Channel\_ID\_Type(Raw\_Channel);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Status, Last);**

**if Raw\_Status in Status\_Type'Pos(Status\_Type'First) .. Status\_Type'Pos(Status\_Type'Last) then**

**Status := Status\_Type'Val(Raw\_Status);**

**else**

**Decode\_Status := Malformed;**

**Status := Status\_Type'First;**

**end if;**

**end if;**

**end Publish\_Reply\_Decode;**

**function Publish\_Result\_Encode**

**(Sender\_Domain : Domain\_ID\_Type;**

**Sender : Module\_ID\_Type;**

**Channel : Channel\_ID\_Type;**

**Message\_Data : CubedOS.Lib.Octet\_Array;**

**Priority : System.Priority := System.Default\_Priority) return Message\_Record**

**is**

**Message : Message\_Record;**

**Request\_ID : Request\_ID\_Type;**

**Position : XDR\_Index\_Type;**

**Last : XDR\_Index\_Type;**

**begin**

**Get\_Next\_Request\_ID(Request\_ID);**

**Message := Make\_Empty\_Message(**

**Sender\_Domain => Sender\_Domain,**

**Receiver\_Domain => Domain\_ID,**

**Sender => Sender,**

**Receiver => ID,**

**Request\_ID => Request\_ID,**

**Message\_ID => Message\_Type'Pos(Publish\_Result),**

**Priority => Priority);**

**Position := 0;**

**XDR.Encode(XDR.XDR\_Unsigned(Channel), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(XDR.XDR\_Unsigned(Message\_Data'Length), Message.Payload, Position, Last);**

**Position := Last + 1;**

**XDR.Encode(Message\_Data, Message.Payload, Position, Last);**

**Position := Last + 1;**

**Message.Size := Position;**

**return Message;**

**end Publish\_Result\_Encode;**

**procedure Publish\_Result\_Decode**

**(Message : in Message\_Record;**

**Channel : out Channel\_ID\_Type;**

**Message\_Data : out CubedOS.Lib.Octet\_Array;**

**Size : out CubedOS.Lib.Octet\_Array\_Count;**

**Decode\_Status : out Message\_Status\_Type)**

**is**

**Position : XDR\_Index\_Type;**

**Raw\_Channel : XDR.XDR\_Unsigned;**

**Raw\_Size : XDR.XDR\_Unsigned;**

**Last : XDR\_Index\_Type;**

**begin**

**pragma Warnings**

**(Off, "unused assignment to ""Last""", Reason => "The last value of Last is not needed");**

**Decode\_Status := Success;**

**Channel := Channel\_ID\_Type'First;**

**Message\_Data := (others => 0);**

**Size := 0;**

**Position := 0;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Channel, Last);**

**Position := Last + 1;**

**if Raw\_Channel in XDR.XDR\_Unsigned(Channel\_ID\_Type'First) .. XDR.XDR\_Unsigned(Channel\_ID\_Type'Last) then**

**Channel := Channel\_ID\_Type(Raw\_Channel);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**if Decode\_Status = Success then**

**XDR.Decode(Message.Payload, Position, Raw\_Size, Last);**

**Position := Last + 1;**

**if Raw\_Size in XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'First) .. XDR.XDR\_Unsigned(CubedOS.Lib.Octet\_Array\_Count'Last) then**

**Size := CubedOS.Lib.Octet\_Array\_Count(Raw\_Size);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**if Size < Message\_Data'Length then**

**XDR.Decode(Message.Payload, Position, Message\_Data(Message\_Data'First .. Message\_Data'First + Size - 1), Last);**

**Decode\_Status := Success;**

**else**

**Decode\_Status := Malformed;**

**end if;**

**end if;**

**end Publish\_Result\_Decode;**

**end CubedOS.Publish\_Subscribe.API;**

**References**

1. <https://en.wikipedia.org/wiki/Scala_(programming_language)>
2. Parr, T. (2012). *The Definitive ANTLR 4 Reference*. The Pragmatic Programmers, LLC. ISBN-13: 978-1-93435-699-9.
3. <https://tools.ietf.org/html/rfc4506.html>
4. <https://tools.ietf.org/html/rfc5531.html>
5. <https://en.wikipedia.org/wiki/Parse_tree>

**Appendix A**

grammar XDR;

@parser::header {

package edu.vtc.xdr2os3;

}

@lexer::header {

package edu.vtc.xdr2os3;

}

@members {

// The following material was from an earlier ANTLR3 grammar. It may not be applicable with

// ANTLR4. Aborting after the first syntax error is not acceptable in a production parser

// anyway. However, I'm keeping this material here for now in case it proves to be a useful

// reference.

// // The following two magic methods, together with the @rulecatch section below cause the

// // parser to exit immediately with an exception when an error is encountered.

// //

// protected Object recoverFromMismatchedToken(IntStream input, int ttype, BitSet follow)

// throws RecognitionException

// {

// throw new MismatchedTokenException(ttype, input);

// }

//

// public Object recoverFromMismatchedSet(IntStream input, RecognitionException e, BitSet follow)

// throws RecognitionException

// {

// throw e;

// }

}

//@parser::rulecatch {

// catch (RecognitionException e) {

// throw e;

// }

//}

/\* ======= \*/

/\* Grammar \*/

/\* ======= \*/

// The productions below are from the XDR standard (RFC-4506). They have been augmented with

// facilities supporting the definition of types with range constraints.

declaration:

type\_specifier IDENTIFIER

| type\_specifier IDENTIFIER EQUALS CONSTANT

| type\_specifier IDENTIFIER LBRACKET value RBRACKET

| type\_specifier IDENTIFIER LANGLE value? RANGLE

| OPAQUE IDENTIFIER LBRACKET value RBRACKET

| OPAQUE IDENTIFIER LANGLE value? RANGLE

| STRING IDENTIFIER LANGLE value? RANGLE

| IDENTIFIER LANGLE value? RANGLE

| type\_specifier STAR IDENTIFIER

| VOID;

subtype\_spec:

IS IDENTIFIER

| IS NATURAL;

value:

CONSTANT | IDENTIFIER;

type\_specifier:

UNSIGNED? INT

| UNSIGNED? HYPER

| FLOAT

| DOUBLE

| QUADRUPLE

| BOOL

| STRING

| IDENTIFIER

| TIME

| TIME\_SPAN

| DATA

| enum\_type\_spec

| struct\_type\_spec

| union\_type\_spec;

enum\_type\_spec:

ENUM enum\_body;

enum\_body:

LBRACE IDENTIFIER EQUALS value (COMMA IDENTIFIER EQUALS value)\* RBRACE

| LBRACE IDENTIFIER (COMMA IDENTIFIER)\* RBRACE;

struct\_type\_spec:

STRUCT struct\_body;

struct\_body:

LBRACE (declaration SEMI)+ RBRACE;

union\_type\_spec:

UNION union\_body;

union\_body:

SWITCH LPARENS declaration RPARENS LBRACE

case\_spec+

(DEFAULT COLON declaration SEMI)?

RBRACE;

case\_spec:

(CASE value COLON)+ declaration SEMI;

constant\_def:

CONST IDENTIFIER EQUALS CONSTANT SEMI

| CONST IDENTIFIER IS IDENTIFIER EQUALS CONSTANT SEMI;

type\_def:

TYPEDEF declaration RANGE range\_constraint SEMI

| TYPEDEF declaration subtype\_spec RANGE range\_constraint SEMI

| TYPEDEF declaration SEMI

| ENUM IDENTIFIER enum\_body SEMI

| STRUCT IDENTIFIER struct\_body SEMI

| UNION IDENTIFIER union\_body SEMI

| MESSAGE STRUCT (LARROW | RARROW) IDENTIFIER struct\_body (condition)? SEMI;

range\_constraint:

CONSTANT DOTDOT CONSTANT

| CONSTANT DOTDOT IDENTIFIER;

line:

declaration SEMI;

definition:

type\_def | constant\_def | encoder\_def | decoder\_def | line;

specification:

definition\*;

// The declarations below are extensions for CubedOS.

encoder\_def:

ENCODER IDENTIFIER LPARENS declaration (COMMA declaration)\* RPARENS SEMI;

decoder\_def:

DECODER IDENTIFIER LPARENS declaration (COMMA declaration)\* RPARENS SEMI;

/\* =========== \*/

/\* Lexer rules \*/

/\* =========== \*/

// --------------

// Reserved Words

// --------------

BOOL : 'bool';

CASE : 'case';

CONST : 'const';

DECODER : 'decoder';

DEFAULT : 'default';

DOUBLE : 'double';

ENCODER : 'encoder';

ENUM : 'enum';

FLOAT : 'float';

HYPER : 'hyper';

INT : 'int';

OPAQUE : 'opaque';

QUADRUPLE : 'quadruple';

RANGE : 'range';

STRING : 'string';

STRUCT : 'struct';

SWITCH : 'switch';

TYPEDEF : 'typedef';

UNION : 'union';

UNSIGNED : 'unsigned';

VOID : 'void';

MESSAGE : 'message';

IS : 'is';

NATURAL : 'Natural';

TIME : 'Ada.Real\_Time.Time';

TIME\_SPAN : 'Ada.Real\_Time.Time\_Span';

DATA : 'CubedOS.Lib.Octet\_Array';

WITH : 'with';

M\_I : 'message\_invariant';

// -------

// Symbols

// -------

COLON : ':';

COMMA : ',';

DOTDOT : '..';

LANGLE : '<';

LBRACE : '{';

LBRACKET : '[';

LPARENS : '(';

EQUALS : '=';

RANGLE : '>';

RBRACE : '}';

RBRACKET : ']';

RPARENS : ')';

SEMI : ';';

STAR : '\*';

LARROW : '<-';

RARROW : '->';

RPOINT : '=>';

LOE : '<=';

GOE : '>=';

IDENTIFIER

: [a-zA-Z][a-zA-Z0-9\_']\*;

condition

: (WITH M\_I RPOINT expression COMMA)\* WITH M\_I RPOINT expression;

expression

: IDENTIFIER LOE IDENTIFIER

| IDENTIFIER GOE IDENTIFIER

| IDENTIFIER RANGLE IDENTIFIER

| IDENTIFIER LANGLE IDENTIFIER

| IDENTIFIER EQUALS IDENTIFIER;

WHITESPACE

: [ \t\f\r\n]+ -> skip;

COMMENT1

: '/\*' .\*? '\*/' -> skip;

COMMENT2

: '//' .\*? [\r\n] -> skip;

CONSTANT

: ('-')? (DECIMAL | BASED) ( ('.') DIGIT+ )? ( ('E' | 'e') DIGIT+ )?;

fragment DECIMAL

: DIGIT ('\_'? DIGIT)\*;

fragment BASED

: DIGIT+ '#' HDIGIT ('\_'? HDIGIT)\* '#';

fragment DIGIT

: [0-9];

fragment HDIGIT

: [0-9a-fA-F];

**Appendix B**