Abstract Syntax Notation One (ASN.1)

Julien Delange <julien.delange@esa.int>

Overview

- Introduction
- Language, syntax
- Tool support, code generation
- Conclusion & perspectives

Julien Delange <julien dot delange at esa dot int>

Overview

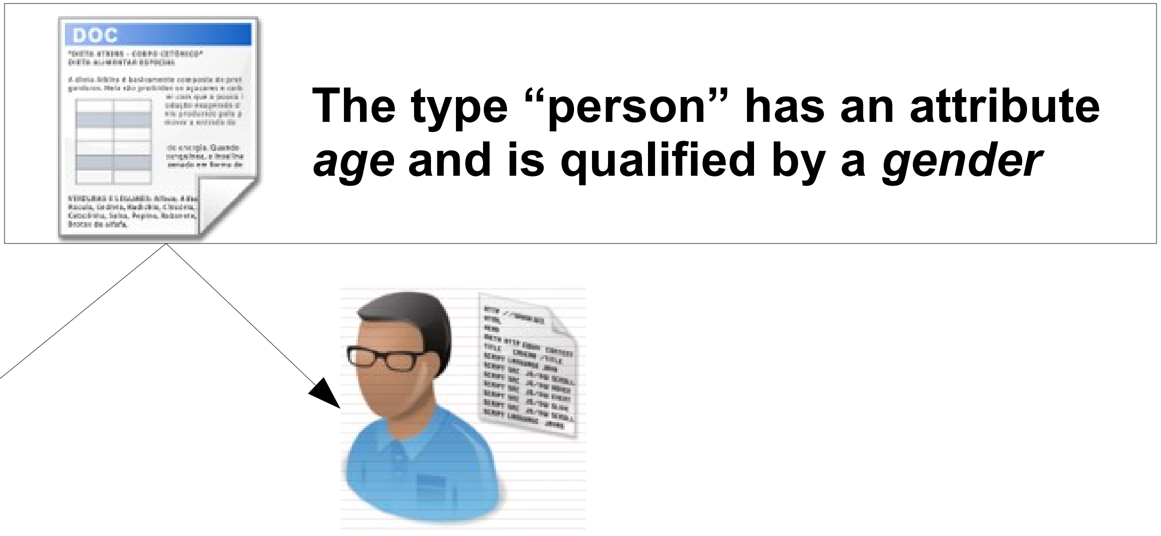
- Introduction
- Language, syntax
- Tool support, code generation
- Conclusion & perspectives

Julien Delange <julien dot delange at esa dot int>

Rationale

- Large projects share data definition
- Need to have a common representation
- Inter-operability between concurrent systems
  - Data representation
  - En/de-coding (marshallers)
- Need to standardize data types

Specification problem



The type “person” has an attribute *age* and is qualified by a *gender*

```
typedef enum {
   Male = 10,
   Female = 20,
   Unknown = 30
}gender_t;

typedef struct
{
   int age;
   genter_t gender;
}person_t;
```

```
typedef struct
{
   int age;
   int gender;
/*
 * 0 = male
 * 1 = female
 * 3 = unknown
 */
}type_person;
```

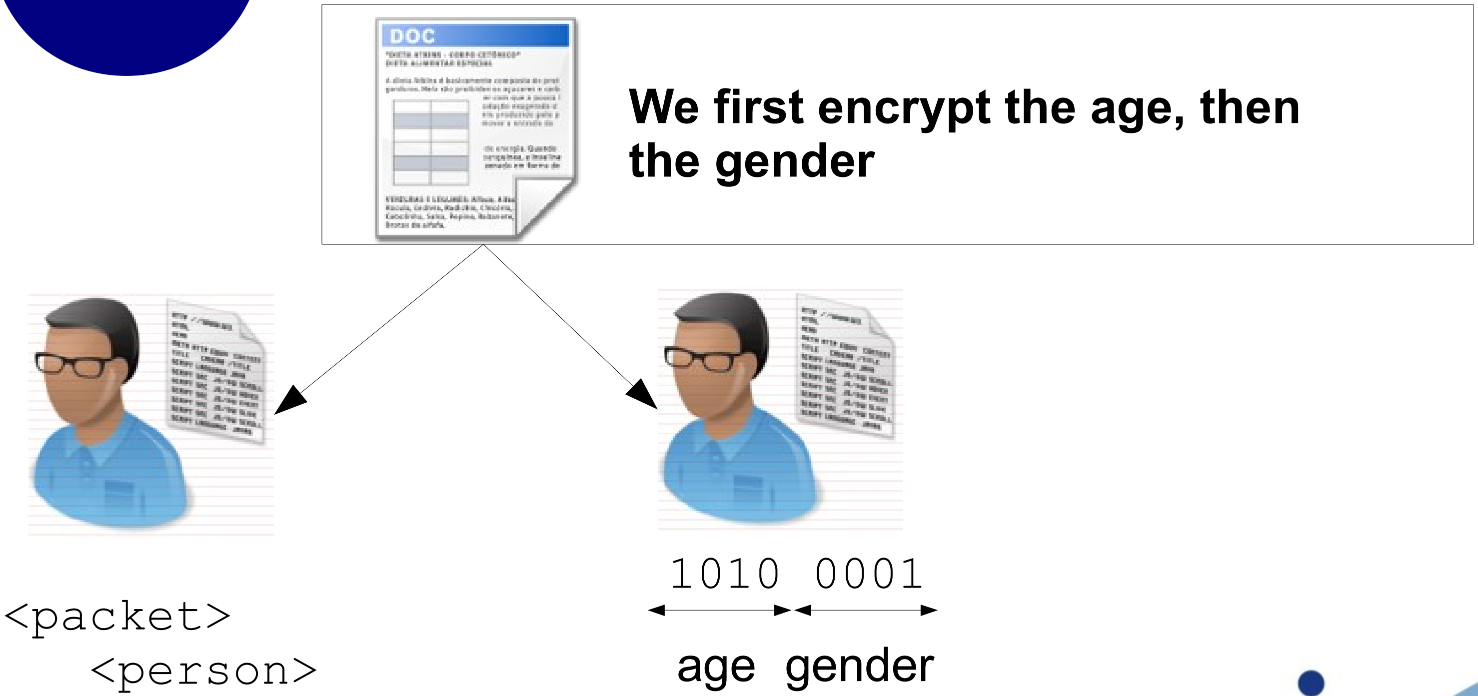
Julien Delange <julien dot delange at esa dot int>

Specification problem

- How to ensure data definition consistency ?
- Data definition for different languages ?
  - Different language, same representation
- How to ensure inter-operability ?
  - Data definition in C used with Ada code ?

Julien Delange <julien dot delange at esa dot int>

Communication/representation problem



We first encrypt the age, then the gender

```
<packet>
   <person>
     <age>10</age>
     <gender>male
     </gender>
</packet>
```

Julien Delange <julien dot delange at esa dot int>

Communication problem

- What is the best method to exchange data ?
  - Textual notation ?
  - Binary ?

- How to ensure application interoperability ?
  - Exchange data across distributed nodes ?

- How to define a common protocol ?

Julien Delange <julien dot delange at esa dot int>

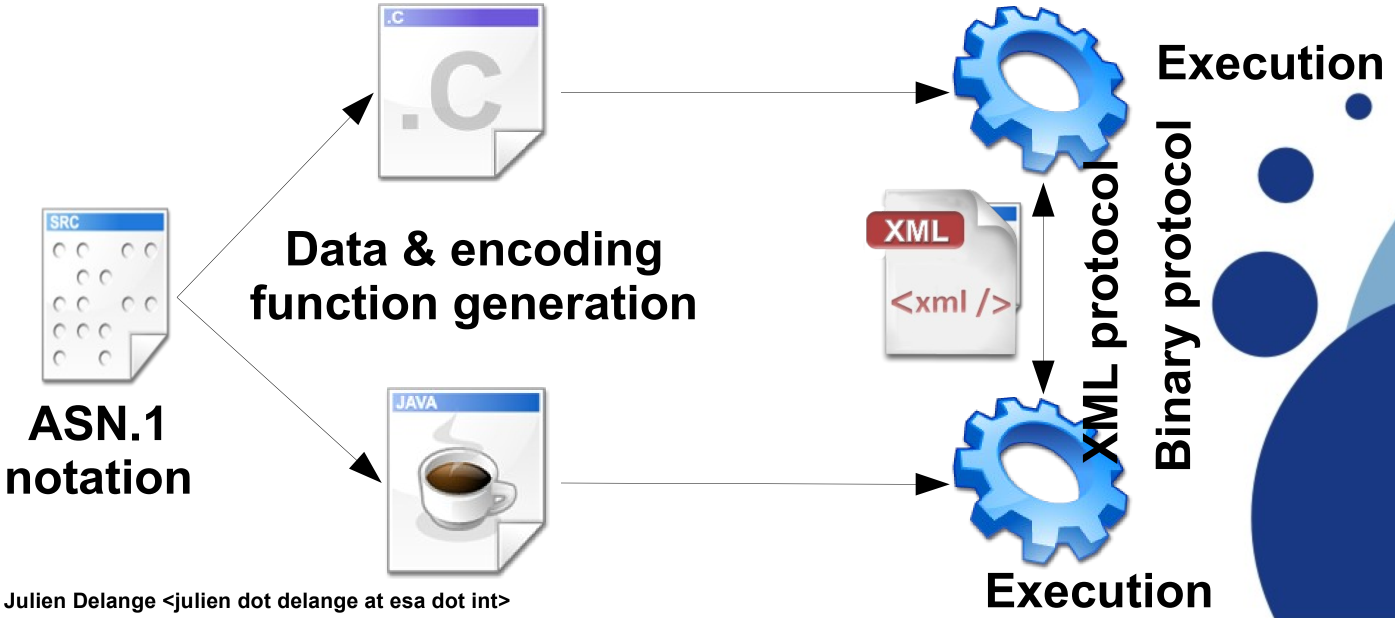
Little bit of history

- Initiated more than 25 years ago !
  - First initiative by ISO/ITU
  - Now, standalone standard (latest revision in 2002)
- Heavily used in telecommunication systems
- Definition of data types & protocols

Julien Delange <julien dot delange at esa dot int>

One notation, several encoding rules

- Data/protocol description with abstract notation
- Generation of encoding functions
  - Standardized communication rules
  - Several encoding rules (binary/xml/...)
  - Bridge the gap with implementation language



Julien Delange <julien dot delange at esa dot int>

Encoding rules (1)

- Basic Encoding Rules (BER)
  - First encoding method
  - LSV encoding rule
- Canonical Encoding Rules (CER)
  - BER improvement
- Distinguished Encoding Rules (DER)

Julien Delange <julien dot delange at esa dot int>

Encoding rules (2)

- XML Encoding Rules (XER)
  - Bridge the GAP with textual exchange (http)
- Packed Encoding Rules (PER)
  - Reduce overhead of BER/CER/DER
- Generic String Encoding Rules (GSER)
  - Human-readable encoding rules

Julien Delange <julien dot delange at esa dot int>

Overview

- Introduction
- Language, syntax
- Tool support, code generation
- Conclusion & perspectives

Julien Delange <julien dot delange at esa dot int>

ASN.1 entities

- Modules
  - Assemble data definitions, separation of concerns
- Types
  - Basic Types & user-Defined types
- Values
  - Related to a defined type
- Identifiers
  - Identify ASN.1 entity

Julien Delange <julien dot delange at esa dot int>

Basic example

Module name : upper case letters !

Type name : upper case for first letters !

Type member : lower case !

```
MyWorld
DEFINITIONS
AUTOMATIC TAGS ::=
BEGIN
   T-Name  ::= UTF8String
   T-Age   ::= INTEGER (0 .. 150)
   T-Person ::= SEQUENCE {
     firstname T-Name DEFAULT “toto”,
     lastname  T-Name DEFAULT “bar”,
     Age       T-Age  DEFAULT 10
   }

T-Small-Group ::=
     SEQUENCE (0 .. 10) of T-Person
   T-Big-Group ::=
     SEQUENCE (0 .. 100) of T-Person
END
```

Julien Delange <julien dot delange at esa dot int>

ASN.1 syntax: comments

- One line comment
  - Begin with --
  - End with -- (or new line)

- Multi-line comment
  - C-style
  - /* ….. */

```
/*
 Define a restaurant module that contains menu
 */
Restaurant DEFINITIONS AUTOMATIC TAGS ::= BEGIN
   T-Starter ::= ENUMERATED {pate, salad}
   -- Define the associated type for a starter course
END
```

Naming rules

```
MODULE MY-SYS DEFINITIONS AUTOMATIC TAGS ::=
TELECOMAND ::= ENUMERATED
{
  move-left     (0),
  move-right    (1),
  move-forward  (2),
  move-backward (3)
}
END
```

- Names
  - Letters/digits/hyphen
  - No lower hyphen !
- Modules
  - Start with an upper-case letter
- Type reference
  - Start with upper case
- Identifiers
  - Start with lower case
- Value reference
  - Start with lower case

Module

- Separation of data types
  - Ease types reuse
- Explicit import/exports

```
MyModule
…
EXPORTS T-Starter, T-Main-Course, T-Dessert;
IMPORT T-Apple, T-Pear FROM FRUITS
       T-Bean FROM VEGETABLES;
…
END
```

First letter: upper case !

```
ModuleName {<object-identifier>}
DEFINITIONS
AUTOMATIC TAGS ::=
BEGIN
   EXPORTS <clauses>
   IMPORTS <clauses>
   <Types definitions>
END
```

Julien Delange <julien dot delange at esa dot int>

ASN.1 basic types

- Numbers
  - Integer (`My-New-Int ::= INTEGER`)
  - Real (`My-New-Real ::= REAL`)
- Boolean
  - Boolean (… `::= SEQUENCE { foobar BOOLEAN,` … `}`)
- String
  - UTF8String (My-Str ::= UTF
  - IA5String
  - Octet String
  - BitString

Julien Delange <julien dot delange at esa dot int>

Types declaration - enumerated & sequence

- Enumerated: types with predefined different values
  - Potentially constrained with values
  - Similar to the enum concept of C

Upper for first letter !

```
T-Starter     ::= ENUMERATED {pate, salad}
T-Main-Course ::= ENUMERATED {meat (3), fish(4)}
T-Dessert     ::= ENUMERATED {icecream, pastry}
```

- Sequence: sequence of one of several values
  - Potentially optional
  - Similar to the struct concept of C

```
T-Complete-Menu ::= SEQUENCE {
     starter    T-Starter OPTIONAL,
     maincourse T-Main-Course,
     dessert    T-Dessert}
T-Big-Meal ::= SEQUENCE (SIZE (2..3))
               OF T-Complete-Menu
```

Julien Delange <julien dot delange at esa dot int>

Types declaration - enumerated & sequence

- Choice
  - One member is included in the type
  - Similar to the enum concept of C

```
T-Starter-Main-Course-Menu ::= SEQUENCE {
    course1 T-Starter,
    course2 T-Main-Course
}

T-Main-Course-Desert-Menu ::= SEQUENCE {
   course1  T-Main-Course,
   course2  T-Desert
}

T-Two-Courses-Menu ::= CHOICE {
   starter-and-main T-Starter-Main-Course-Menu,
   main-and-desert  T-Main-Course-Dessert-Menu
}
-- Define a value for the most served meal
most-served-two-course T-Main-Course-Desert-Menu
              ::= {course1 meat, course2 pastry}
```

Julien Delange <julien dot delange at esa dot int>

Types constraints

- Range of numeric types

```
T-Rotation  ::= REAL (-180.0 .. 180.0)
T-Minute    ::= INTEGER (0 .. 59)
T-No-Zero   ::= INTEGER (ALL EXCEPT 0)
T-Note      ::= (0..20 | 42)
```

- Size of string

```
T-Content ::= OCTET STRING (SIZE(0..128))
```

- Default value

```
T-Starter-Main-Course-Menu ::= SEQUENCE {
    course1 T-Starter DEFAULT pate,
    course2 T-Main-Course DEFAULT meat
}
```

Value definition

```
Notation DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
   T-Note ::= INTEGER (0..20 | 42)
   best-note T-Note ::= 42
   worst-note T-Note ::= 0
END
```

- `<value> <asn1-type> ::= <assigned_value>`

- From the example
  - Note is a value between 0 and 20, or 42
  - New values in target language
    - `best_note`
    - `worst_note`

Julien Delange <julien dot delange at esa dot int>

Overview

- Introduction
- Language, syntax
- Tool support, code generation
- Conclusion & perspectives

Julien Delange <julien dot delange at esa dot int>

Tool support: asn1c

- Open-Source, BSD-licensed
- Limited language support
    - C/C++
- Support for different encoding method
    - BER/DER/XER/PER
- cf. http://lionet.info/asn1c/blog/

Asn1 Space Certifiable Compiler (ASN1Scc)

- Dual-license
  - Free for non-commercial projects

- Targets safety-critical systems
  - Support of Ada/C
  - Static allocation of all resources

- Support several encoding method
  - BER/XER/PER

- cf. http://www.semantix.gr

A closer look to ASN1Scc

- Open code generation
  - No glob nor arch-dependent code
  - Potential to explore generated types
- Static allocation
  - Fit with safety-critical requirements
  - Avoid indeterminism
- Rely on predefined encoding functions

Julien Delange <julien dot delange at esa dot int>

Generated functions definition

```
My-Integer ::= INTEGER (0 .. 65535)
```

```
typedef asn1SccSint asn1SccMy_Integer;
#define asn1SccMy_Integer_REQUIRED_BYTES_FOR_ENCODING       2
#define asn1SccMy_Integer_REQUIRED_BITS_FOR_ENCODING       16
#define asn1SccMy_Integer_REQUIRED_BYTES_FOR_ACN_ENCODING   2
#define asn1SccMy_Integer_REQUIRED_BITS_FOR_ACN_ENCODING   16
#ifndef ERR_asn1SccMy_Integer
#define ERR_asn1SccMy_Integer    1001 /* (0..65535) */
#endif
void asn1SccMy_Integer_Initialize
                   (asn1SccMy_Integer* pVal);
flag asn1SccMy_Integer_IsConstraintValid
                   (const asn1SccMy_Integer* val,
                    int* pErrCode);
flag asn1SccMy_Integer_Equal
                   (const asn1SccMy_Integer* val1,
                    const asn1SccMy_Integer* val2);
flag asn1SccMy_Integer_Encode
                   (const asn1SccMy_Integer* val,
                    BitStream* pBitStrm,
                    int* pErrCode,
                    flag bCheckConstraints);
flag asn1SccMy_Integer_Decode(asn1SccMy_Integer* pVal,
                              BitStream* pBitStrm,
                              int* pErrCode);
```

Basic types manipulation

```
My-Integer ::= INTEGER (0 .. 65535)
```

```
void My_Integer_Initialize(My_Integer* pVal)
{
    *pVal = 0;
}

flag My_Integer_IsConstraintValid(const My_Integer* pVal, int* pErrCode)
{
    if ( !(((*pVal>=0) && (*pVal<=65535))) ) {
        *pErrCode = ERR_My_Integer;
        return FALSE;
    }
    (void)pVal;
    (void)pErrCode;
    return TRUE;
}

flag My_Integer_Equal(const My_Integer* pVal1,
                      const My_Integer* pVal2)
{
    if (*pVal1 != *pVal2)
        return FALSE;
    return TRUE;
}
```

Julien Delange <julien dot delange at esa dot int>

Basic encoding functions generation

```
My-Integer ::= INTEGER (0 .. 65535)
```

```
flag Keycode_Encode(const Keycode* pVal,
                    BitStream* pBitStrm,
                    int* pErrCode,
                    flag bCheckConstraints)
{

    if (bCheckConstraints && !Keycode_IsConstraintValid(pVal, pErrCode))
        return FALSE;
    BitStream_EncodeConstraintWholeNumber(pBitStrm, *pVal, 0, 65535);
    return TRUE;
}

flag Keycode_Decode(Keycode* pVal,
                    BitStream* pBitStrm,
                    int* pErrCode)
{

    if (!BitStream_DecodeConstraintWholeNumber
                  (pBitStrm, pVal, 0, 65535)) {
        *pErrCode = ERR_INSUFFICIENT_DATA;
        return FALSE;
    }
    return TRUE;
}
```

Closer look to initialization

```
Geometry DEFINITIONS AUTOMATIC TAGS ::= BEGIN

T-Rectangle ::= SEQUENCE {
  length INTEGER DEFAULT 10,
  width  INTEGER DEFAULT 20
}

END
```

```
void T_Rectangle_Initialize
            (T_Rectangle* pVal)
{
    pVal->length = 10;
    pVal->width = 20;
}
```

```
typedef struct {
    asn1SccSint length;
    asn1SccSint width;
    struct {
        unsigned int length:1;
        unsigned int width:1;
    } exist;
} T_Rectangle;
```

Julien Delange <julien dot delange at esa dot int>

Closer look to comparison

```
Geometry DEFINITIONS AUTOMATIC TAGS ::= BEGIN

T-Rectangle ::= SEQUENCE {
  length INTEGER DEFAULT 10,
  width  INTEGER DEFAULT 20
}

END
```

```
flag T_Rectangle_Equal(const T_Rectangle* pVal1,
                       const T_Rectangle* pVal2)
{
     if (pVal1->exist.length !=
         pVal2->exist.length)
          return FALSE;
     if (pVal1->exist.length) {
          if (pVal1->length !=
              pVal2->length)
               return FALSE;
     }
     if (pVal1->exist.width !=
         pVal2->exist.width)
          return FALSE;
     if (pVal1->exist.width) {
          if (pVal1->width != pVal2->width)
               return FALSE;
     }
     return TRUE;
}
```

```
typedef struct {
    asn1SccSint length;
    asn1SccSint width;
    struct {
        unsigned int length:1;
        unsigned int width:1;
    } exist;
} T_Rectangle;
```

Closer look to constraints validation

```
Geometry DEFINITIONS AUTOMATIC TAGS ::= BEGIN

T-Rectangle ::= SEQUENCE {
  length INTEGER DEFAULT 10,
  width  INTEGER DEFAULT 20
}

END
```

```
typedef struct {
    asn1SccSint length;
    asn1SccSint width;
    struct {
        unsigned int length:1;
        unsigned int width:1;
    } exist;
} T_Rectangle;
```

```
flag T_Rectangle_IsConstraintValid
      (const T_Rectangle* pVal, int* pErrCode)
{
     if (pVal->exist.length) {
     }
     if (pVal->exist.width) {
     }
     (void)pVal;
     (void)pErrCode;
     return TRUE;
}
```

Julien Delange <julien dot delange at esa dot int>

Encoding for enumerated types

`T-Main-Course ::= ENUMERATED {`**`meat (3), fish(4)`**`}`

```
flag T_Main_Course_Encode(const T_Main_Course* pVal,
                          BitStream* pBitStrm,
                          int* pErrCode,
                          flag bCheckConstraints)
{
     if (bCheckConstraints &&
         !T_Main_Course_IsConstraintValid
                        (pVal, pErrCode))
          return FALSE;
     switch(*pVal)
     {
          case meat:
               BitStream_EncodeConstraintWholeNumber
                 (pBitStrm, 0, 0, 1);
               Break;
        case fish:
               BitStream_EncodeConstraintWholeNumber
                 (pBitStrm, 1, 0, 1);
               break;
          default:
               *pErrCode =
          ERR_T_Main_Course_unknown_enumeration_value;
               return FALSE;
     }
     return TRUE;
}
```

No encoding of the type value

```
typedef enum {
    meat = 3,
    fish = 4
} T_Main_Course;
```

Julien Delange <julien dot delange at esa dot int>

Decoding for enumerated types

```
T-Main-Course ::= ENUMERATED {meat (3), fish(4)}
```

```
flag T_Main_Course_Decode(T_Main_Course* pVal,
                          BitStream* pBitStrm,
                          int* pErrCode)
{
     asn1SccSint enumIndex = 0;

     if (!BitStream_DecodeConstraintWholeNumb
         (pBitStrm, &enumIndex, 0, 1)) {
          *pErrCode = ERR_INSUFFICIENT_DATA;
          return FALSE;
     }
     switch(enumIndex)
     {    case 0:
               *pVal = meat;
               break;
          case 1:
               *pVal = fish;
               break;
          default:
               *pErrCode =
         ERR_T_Main_Course_unknown_enumeration_value;
               return FALSE;
     }
     return TRUE;
}
```

```
typedef enum {
    meat = 3,
    fish = 4
} T_Main_Course;
```

Restitution of the ASN.1 defined value

Julien Delange <julien dot delange at esa dot int>

Use generated code from ASN1Scc

`My-Type ::= <type_definition>`

- Generation of new type
  - **Generation pattern: `ASN1_Name` (ex: `My_Type`)**
  - **Define in `<module.h>` generated header file**

- Type creation and validation functions
  - **`<ASN1_Name>_Is_Constraint_Valid (<ASN1_Name>*)`**
  - **`<ASN1_Name>_Initialize (<ASN1_Name>*)`**
  - **`<ASN1_Name>_Equal (<ASN1_Name>*, <ASN1_Name>*)`**

- Encoding/decoding to/from bit streams
  - **`<ASN1_Name>_Encode (<ASN1_Name>*, BitStream*, int*, flag)`**
  - **`<ASN1_Name>_Decode (<ASN1_Name>*, BitStream*, int*)`**

Generated code usage: example

```
Geometry DEFINITIONS AUTOMATIC TAGS ::= BEGIN
T-Rectangle ::= SEQUENCE {
  length INTEGER DEFAULT 10,
  width  INTEGER DEFAULT 20
}
END
```

User code

```
#include <geometry.h>
int main()
{
   T_Rectangle rect1, rect2;
   T_Rectangle_Initialize (&rect1);
   T_Rectangle_Initialize (&rect2);
   rect1.length = 3; rect1.width = 4;
   rect1.exist.width = 1; rect1.exist.length = 1;
   rect2.length = 3; rect2.width = 5;
   rect2.exist.width = 1; rect2.exist.length = 1;
   T_Rectangle_Equal (&rect1, &rect2) ?
       printf ("Egaux\n") : printf ("Differents\n");
   rect2.width = 4;
   T_Rectangle_Equal (&rect1, &rect2) ?
       printf ("Egaux\n") : printf ("Differents\n");
   return (0);
}
```

Generated type definition

```
typedef struct {
    asn1SccSint length;
    asn1SccSint width;
    struct {
        unsigned int length:1;
        unsigned int width:1;
    } exist;
} T_Rectangle;
```

Julien Delange <julien dot delange at esa dot int>

Generated code usage: example (2)

```
Geometry DEFINITIONS AUTOMATIC TAGS ::= BEGIN
T-Rectangle ::= SEQUENCE {
  length INTEGER DEFAULT 10,
  width  INTEGER DEFAULT 20
}
END
```

User code

```
#include <geometry.h>
#include <string.h>
#include <stdio.h>

int main()
{
   T_Rectangle rect1, rect2;
   BitStream bs1, bs2;
   char buf1[T_Rectangle_REQUIRED_BYTES_FOR_ENCODING];
   char buf2[T_Rectangle_REQUIRED_BYTES_FOR_ENCODING];
   int errcode;
   BitStream_Init(&bs1, buf1, T_Rectangle_REQUIRED_BYTES_FOR_ENCODING);
   T_Rectangle_Initialize (&rect1);
   rect1.length = 3; rect1.width = 4; rect1.exist.width = 1;
   rect1.exist.length = 1;

   printf ("rect1 length=%lld; width=%lld\n", rect1.length, rect1.width);
   if (! T_Rectangle_Encode(&rect1, &bs1, &errcode, 1))
   {
      printf ("Encoding error : %d\n", errcode);
   }
   BitStream_Init(&bs2, buf2, T_Rectangle_REQUIRED_BYTES_FOR_ENCODING);
   memcpy (buf2, buf1, T_Rectangle_REQUIRED_BYTES_FOR_ENCODING);
   if (! T_Rectangle_Decode(&rect2, &bs2, &errcode))
   {
      printf ("Decoding error: %d\n", errcode);
   }
   printf ("rect2 length=%lld; width=%lld\n", rect2.length, rect2.width);
   return (0);
}
```

Generated type definition

```
typedef struct {
    asn1SccSint length;
    asn1SccSint width;
    struct {
        unsigned int length:1;
        unsigned int width:1;
    } exist;
} T_Rectangle;
```

Julien Delange <julien dot delange at esa dot int>

Generated code usage: example (3)

```
T-Main-Course-Desert-Menu ::= SEQUENCE {
   course1  T-Main-Course,
   course2  T-Desert
}
T-Main-Course ::= ENUMERATED {meat (3), fish(4)}
T-Desert      ::= ENUMERATED {icecream, pastry}
```

Generated types definition

```
typedef enum {
    meat = 3,
    fish = 4
} T_Main_Course;
```

```
typedef enum {
    icecream = 0,
    pastry = 1
} T_Desert;
```

```
typedef struct {
    T_Main_Course course1;
    T_Desert course2;
} T_Main_Course_Desert_Menu;
```

Generated values definition

```
T_Main_Course_Desert_Menu most_served_two_course =
{
      .course1 = meat,
      .course2 = pastry
};
```

User code

```
#include <stdio.h>
#include <restaurant.h>

void print_main_and_desert_menu (T_Main_Course_Desert_Menu* val)
{
   printf ("Main course: %s\n", (val->course1 == meat) ? "meat" : "fish");
   printf ("Main course: %s\n", (val->course2 == pastry) ? "pastry" :
"icecream");
}

int main ()
{
   T_Main_Course_Desert_Menu menu = most_served_two_course;
   print_main_and_desert_menu (&most_served_two_course);
   return 1;
}
```

Julien Delange <julien dot delange at esa dot int>

Overview

- Introduction
- Language, syntax
- Tool support, code generation
- Conclusion & perspectives

Julien Delange <julien dot delange at esa dot int>

Conclusion

- Standardized method for types definition

- Ensure protocols & implementation consistency
  - Use of different languages
  - Automatic generation of encryption functions

- Avoid traditional programming pitfalls
  - Interpretation of data types
  - Errors in protocols implementation

- Use it !

Julien Delange <julien dot delange at esa dot int>

Resources

- United Nation Agency for Information and Communication technology issues: http://itu.int
- Learn
  - ASN.1 reference card: http://www.oss.com/asn1/tutorial/ReferenceCard.html
  - ASN1 books available on the internet
    - http://www.oss.com/asn1/larmouth.html
    - http://www.oss.com/asn1/dubuisson.html
- Tools
  - ASN1C: http://lionet.info/asn1c/compiler.html
  - ASN1SCC: http://www.semantix.gr