

WHITE PAPER

**Develop your own ASN.1 solution based on an open
XML architecture**

(Version 1.1 - December 2001)

Atos 
Origin

1. ABSTRACT

With its new ASN.1 product line (**MARBEN™** ASNSDK), Atos Origin provides an innovative solution to satisfy users' specific needs such as building ASN.1 to other formal notation gateways, building generic decoding engine or many other applications. This solution mainly rest on the generation of an XML Semantic Tree, which represents a synthesis of all the information, contained in the input ASN.1 description. By specifying the wanted conversion or generation rules through an XSLT (eXtensible Stylesheet Language Transformation) file and thanks to the use of common XSL processor, the user may easily build a gateway to another formal notation (XML, CSN.1...) or its own ASN.1 encoding/decoding engine satisfying his specific requirements. This added-value feature can be of particular interest for users who develop generic decoding engines such as for instance protocol analyzer suppliers, generic platform services or Network supervisors.

2. WHAT ARE THE NEEDS TO SATISFY

Most of ASN.1 tools are built on the same architecture principles (see fig.1). They are composed of an ASN.1 compiler and ASN.1 (BER, DER and PER) encoding/decoding libraries. Based on the input ASN.1 description, the ASN.1 compiler generates an application programming interface (more commonly data structure in C, C++ or Java) and either tables or code which the ASN.1 encoding/decoding run-times have to use to perform encode/decode operations. User code shall be developed above the generated API, compiled with either the table or code generated, then the whole shall be linked with encoding/decoding run-time library to constitute the final executable.

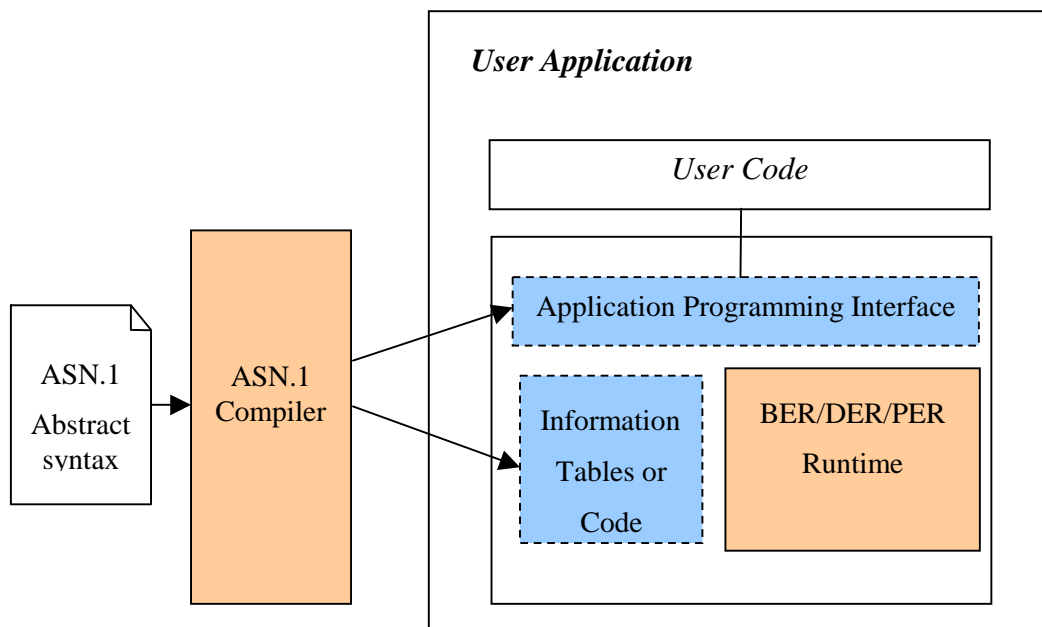


Figure 1. ASN.1 tools architecture principles

Although those tools give the capital advantage to take benefit from fully tested and ready to use encoding/decoding engines, they have the following drawbacks:

- The C, C++ or Java API provided by those tools are fixed. They are generated based on hard-coded rules (sometimes slightly tunable) which give for each ASN.1 production the corresponding programming language data structure to be generated. Those interfaces do sometimes not satisfy the user needs and require the user to do some conversion from the generated API data structures to the expected application data structures. Such an operation with no functional added value is CPU time and memory consuming, and considerably degrades the performances of the whole application. For high constraint or real-time applications with high-level performance requirements, it can even be unacceptable.
- Each addition or evolution in the input ASN.1 description, as for instance new definitions of types, will directly modify the API generated by those tools. That API modification compulsory implies a new compilation of the user application. You will say that it is not a major drawback since when new definitions are added in the ASN.1 description, the user application will have necessary to be modified and re-compiled in order to process them. In most of the cases, you are probably right. But take for instance applications such as Network Supervisors or protocol analyzers. The key issue for those applications is to display (through a graphical interface) information in a convivial manner for the end-user. These applications must be generic; meaning they shall support all the versions of all the ASN.1 descriptions possibly used by network equipment. At this point you will understand that an ASN.1 generic-decoding engine (with a generic API) represents a capital advantage: No need to recompile the entire application to support a new version of an ASN.1 specification, no need to manage different versions of those applications, no need to set up heavy and costly procedures to update numerous equipment already deployed in the field by customers. The update procedure comes down to a data file downloading which contains all the information of the new ASN.1 description version.
- Those tools are designed on closed architecture. The ASN.1 compiler and the different encoding/decoding engine libraries are imperatively tied. You cannot use one ASN.1 compiler from a particular supplier in conjunction with an encoding/decoding engine from another one. All the information generated by the ASN.1 compiler are proprietary. The API is still documented but the encoding/decoding information table (containing all the information used by the encoding/decoding engine to encode/decode values) are never documented. Once again, you could think, “Why should I care of those tables? What would be the interest to access this incomprehensible information?” We understand your issue. People involved in the [Z39.50](#) (Information Retrieval Protocol) standardization process or who have already thought of building gateways between ASN.1 to other formal notation are catching the point. Imagine that instead of generating information table in a specific programming language following a proprietary format, an ASN.1 compiler generates an XML format file, which represents a synthesis of the input ASN.1 description, fully documented, with an associated XML Schema. Then, you just have to provide an XSLT file, pass it in a common XSL processor (such as those from Apache or Sun) and you will have your gateway to another formal notation (XML, CSN.1,...) without writing any code!

In addition to a complete set of fully tested and ready to use ASN.1 encoding/decoding engines, **MARBEN™** ASNSDK product provides innovative solution to satisfy the above mentioned requirements. This solution mainly rest on the generation of an XML Semantic Tree, which represents a synthesis of the input ASN.1 description in an XML, format. From this XML Semantic Tree, you can easily parse and retrieve all the information required to develop your own ASN.1 encoding/decoding engine or to build gateways to other formal notation (CSN.1, XML,).

3. XML SEMANTIC TREE

3.1 ARCHITECTURE OVERVIEW

In addition to a syntactic and semantic analyzer, the ASNSDK ASN.1 compiler encompasses an XML Semantic Tree generator. The generated XML Semantic Tree represents the synthesis of all the information of the input ASN.1 description in an XML format.

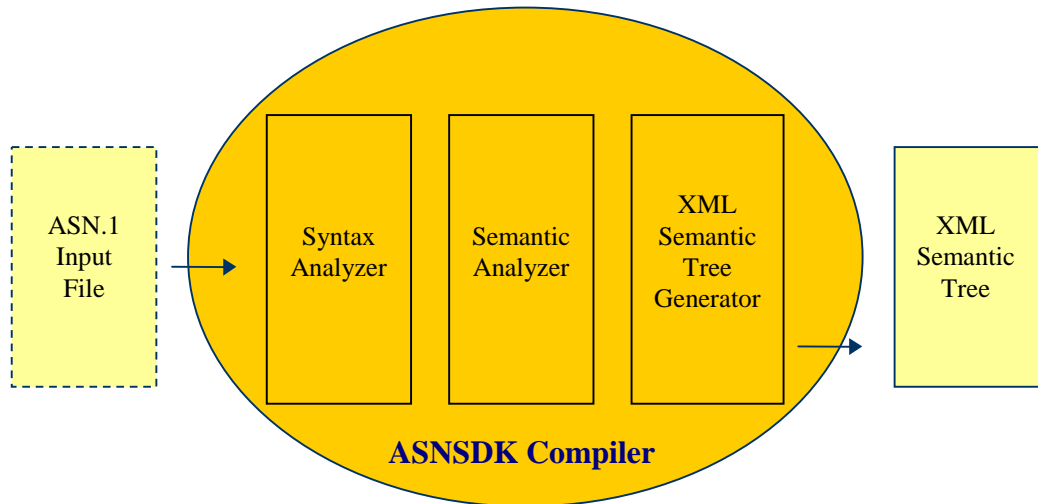


Figure n°2: XML Semantic Tree Generator

3.2 DESCRIPTION

The structure of this XML Semantic Tree is fully described by the associated XML schema and an associated UML model. The information contained in the input ASN.1 description is represented following a tree structure. The basic principle consists in generating an XML element for each ASN.1 type definition.

For each ASN.1 type and so XML element, data related to the PER, BER or DER encoding/decoding of values of those types are computed. Such data consist in BER Tags, DER components orders, PER effective constraints and size (number of bits) of each encoding field,...

In order to ease the data processing, lexical details of the ASN.1 description are hidden to the user. For instance, all the IMPORT and EXPORT clauses are resolved, and information object sets are translated into their associated tables.

3.3 EXAMPLE

Let's take as example a very simple ASN.1 description:

```

Example DEFINITIONS AUTOMATIC TAGS ::=
BEGIN
  
```

```
MyType ::= SEQUENCE
{
  fieldA INTEGER (0..15),
  fieldB BOOLEAN
}

END
```

From that ASN.1 description, the ASNSDK Compiler generates the following XML semantic Tree. (...) means that non useful information have been removed from the XML semantic Tree generated.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
Copyright (C) 2000, 2001 Atos Origin. All rights reserved.

(...)

-->
<GeneTypeTable (...) >
  <TypeAssignmentList>
    <TypeAssignment (...) typeReference="MyType"
      typeIdRef="t2" (...)/>
  </TypeAssignmentList>
  <TypeList>
    <Boolean typeId="t0"/>
    <Integer typeId="t1">
      <PerEffectiveNumberConstraint lb="0" ub="15" (...) />
    </Integer>
    <Sequence typeId="t2" typeReference="MyType">
      (...)
      <SequenceRoot1>
        <SetSeqNamedType identifier="fieldA" (...)
          typeIdRef="t1"
          (...) />
        <SetSeqNamedType identifier="fieldB" fieldNumber="1"
          typeIdRef="t0"
          (...) />
      </SequenceRoot1>
    </Sequence>
  </TypeList>
</GeneTypeTable>
```

4. SHOW CASE: BUILDING A UMTS PROTOCOL ANALYZER

UMTS Radio Access Network (RAN) protocols (Iu, Iub, Iur, and RRC) are all described in ASN.1 using PER encoding rules to save bandwidth when transmitting data. 3GPP organization, responsible for the standardization process of UMTS, has defined three versions of UMTS specifications named release 99, Release 4 and Release5. Within a given release (as for instance release 99 which is the one currently developed by telecom equipment manufacturers), many different versions have been issued and will be issued. They impact the ASN.1 descriptions of RAN protocols by adding new definitions or corrections. The frequency of publications is around one every three months.

In this quickly evolving process, you can easily imagine what is the big issue that UMTS RAN protocol analyzer suppliers are facing. Every three months, they shall take into account a new version of ASN.1 descriptions, modify their applications to process the new feature and corrections performed, updates all the equipment already deployed in the fields by numerous customers. And of course, all these actions must be done in a very short notice to be time to market and to answer to customers needs. Moreover, all previous versions must still be supported, as all customers do not switch on the latest version at the same time. In few words: a true nightmare.

Taking into account this whole problematic, it is obvious that the design of a generic application supporting all versions of all ASN.1 descriptions that may be used by network equipment will give a capital advantage to such UMTS RAN protocol analyzer by saving time and money. At this stage you may think that it would be marvelous but it is only a wish. Not at all!

As the aim of a protocol analyzer is only to decode and display data in a convivial manner (with the possible definitions of filters on such or such message fields), you can build such a generic application which can decode, filter and display all the messages of a new ASN.1 description without being obliged to be re-compiled and linked. This solution mainly relies on the development of a PER generic decoding engine composed of an ASN.1 information loader and a decoding engine (see figure n°3).

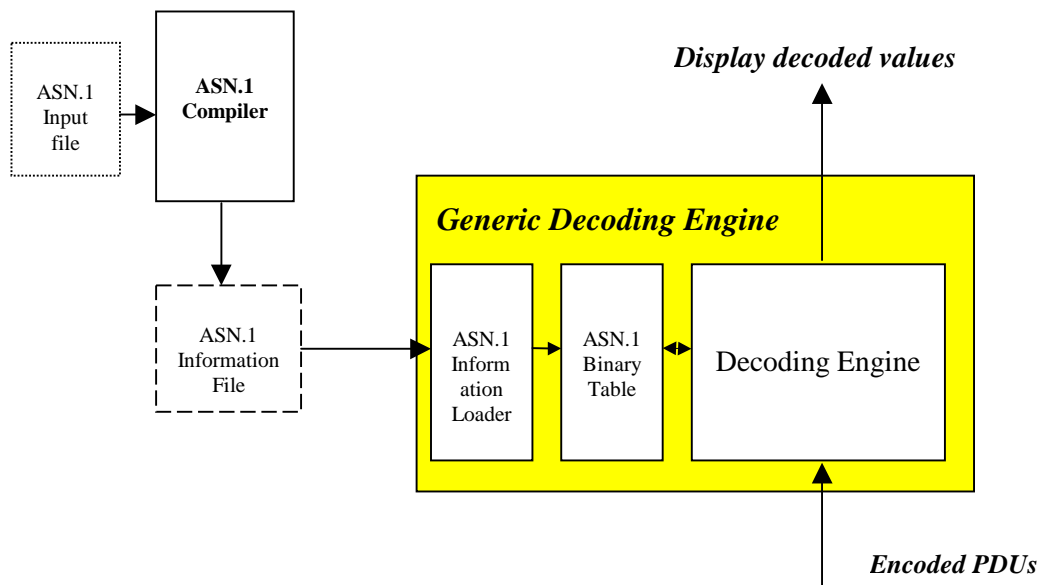


Figure n°3: Generic Decoding Engine architecture overview

The ASN.1 information loader is responsible for translating the information generated by an ASN.1 compiler and reflecting the input ASN.1 description in a binary form directly usable by the decoding engine. With such architecture, when a new ASN.1 description shall be supported, the update procedure comes down to a data file download, which contains all the information of the new ASN.1 description.

Atos Origin ASNSDK product provides you the means to build such a generic application. As mentioned in the previous chapter, ASNSDK ASN.1 compiler generates an XML Semantic Tree, which represents in an XML format, a synthesis of the input ASN.1 description. This XML Semantic Tree represents the ASN.1 Information File described in figure n°3. However, instead of being in a specific programming language following a proprietary format, it has the capital advantage to be in XML format fully documented with an associated XML Schema. Thus, you can easily build an ASN.1 Information Loader just by defining an XSL file, specifying the translation rules that shall be applied on the XML Semantic Tree to produce the binary tables you want and passing it to an XSL processor (such as those from Apache or Sun).

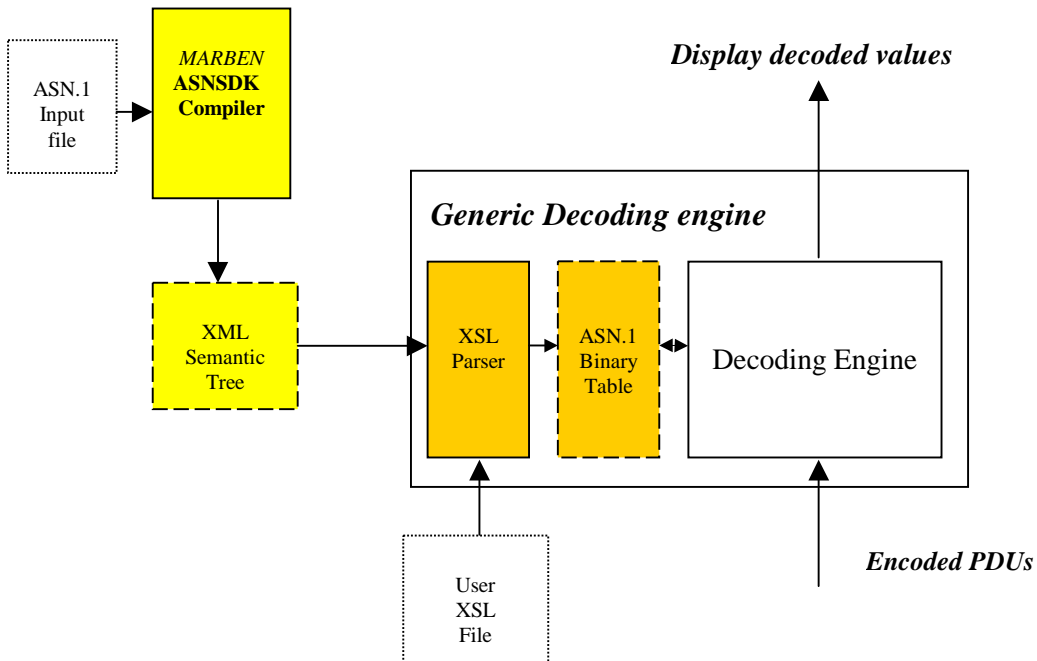


Figure n°4: Generic Decoding Engine using XML Semantic Tree

By using our XML Semantic Tree feature, you save time and effort in the processing of the ASN.1 description (syntactic, semantic checks, representation of ASN.1 encoding/decoding information) to directly focus on the design and the architecture of your application, which represents the true added value of your solution.

5. COMING SOON

Based on this key feature of our open ASN.1 tools architecture, we are currently developing added value services:

- An XML encoding/decoding engine conformant to the next coming XER (X.693) standard,
- An XML Schema Generator that will generate, from input ASN.1 description, an XML Schema that conforms to XER.
- Default XML Style sheet generator.

This whole set of features will give our customers the capability to :

- encode any XML value, conform to the XML schema generated, in BER or PER encoding, thus providing an “XML interface” to ASN.1 BER and PER encoding/decoding engine,
- use common XML tools such as XML Spy to edit and modify ASN.1 values,
- use common XML browsers such as Internet Explorer or Netscape to display ASN.1 values in the way you want by just modifying the default XML style sheet generated.

Our aim is to provide our customers with integrated solutions and tools taking benefit of both ASN.1 and XML technologies.

For more information, visit our web site <http://www.marben-products.com> or contact us at support@marben-products.com.

INFORMATION

sales@marben-products.com

support@marben-products.com

ABOUT US

Atos Origin is acknowledged as a leader in Telecommunication software solutions. World-class communications solutions (OSI, SONET/SDH, IP) are marketed under its *MARBEN™* PRODUCTS line. Atos Origin has extensive experience in porting its software technology to a wide variety of platforms ranging from mainframe and desktop systems to small embedded computers. The *MARBEN™* PRODUCTS line brings suitable response to each type of needs.

To get more information, visit Atos Origin website at <http://www.marben-products.com>

MARBEN™ PRODUCTS is a trademark of Atos Origin. All other names are used for identification purposes only and are trademarks or registered trademarks of their respective companies.