ASN1C ASN.1 Compiler Training

NL Telecom Co., Ltd.

Version 1.4c

2011-5-12

ASN1C/ASN1CPP Training

차 례

1. ASN.1 의 개요 및 필요성

1-1. ASN.1 개요

ASN.1 은 Abstract Syntax Notation One 을 말하며, 국제전기통신 연합(ITU)에서 정의한 네트웍상의 데이터 교환을 정의한 프로토콜로 X.208, X.209 및 X.690 에 정의되어 있습니다.

OSI 참조 모델에서 ASN.1 은 네트워크 관리 시스템에 있는 관리되는 개체와 같이 데이터 Structure 를 기술하는데 사용되는 표시법입니다.

ASN.1 은 머신 독립적이고 많은 네트워크 콘텍스트에서 사용됩니다. 예를 들면 OSI 네트워크 관리 프레임워크와 인터넷 TCP/IP 프로토콜 모음에서 온 SNMP 양쪽 모두에서와 같이 어플리케이션층의 패킷을 기술하는데 사용됩니다.

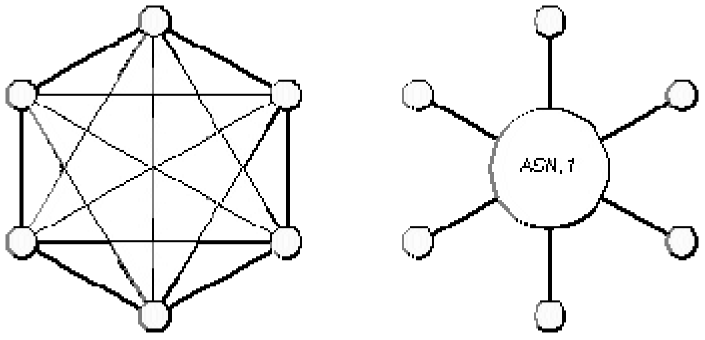
ASN.1 은 각 끝점에서 다른 변조 시스템을 사용할 수 있는 ES(End System) 사이의 정보를 전송하기 위한 공동 Syntax 로서 기능을 한다.

네트워크상에 존재하는 다양한 종류의 시스템들은 각각 데이터를 표현하는 독특한 방식을 가지고 있습니다. 따라서 네트워크 상에서 메시지를 교환하기 위해서는 모든 시스템에서 받아들일 수 있는 형태의 호환성 있는 데이터 표현방식을 정의할 필요가 있습니다. 이러한 제반 환경에 의해서 ASN.1 은 정의되게 되었다. ASN.1 에서는 INTEGER 나 각종 STRING 형태의 데이터를 비롯하여 그러한 ASN.1 content 들의 모임인 SEQUENCE 나 SET 등의 데이터 표현방식을 정의함으로써 대부분의 네트워크 상에서 교환되어지는 메시지들을 ASN.1 방식에 따라서 표현할 수 있도록 하였습니다.

1-2. ASN.1 필요성

1-2-1. System 간의 차이

Sun 과 Windows 간의 Data 를 주고 받는 Application 을 작성해본 사람들은 알겠지만 동일한 Data 를 주고 받으면 Computer System 에 따라 분석하는 것이 다르기 때문에 엉뚱한 값을 받게 된다. 예를 들어, 100 이라는 값을 Sun 에서 Windows 시스템으로 동일한 32 bit integer 를 사용하여 보내면 Windows 에서는 매우 큰 수로 인식하게 된다. (1,677,721,600) 이를 전문 용어로 Endian 이라 하는데, Sun 에서는 Motorola 계열의 Big Endian 을 사용하여 100 을 [00 00 00 64]와 같이 인식한다. 이와는 반대로 Intel 진영인 Little Endian 의 Windows 에서는 100 을 [64 00 00 00]와 같이 인식한다. 따라서 어느 한쪽에서 Byte 단위로 Swapping 을 해주어야 한다.
이는 물론 Integer 가 32 bit 인 경우에 해당 되는 것이고 64-bit machine 이나 8-bit, 16-bit 인 경우에는 또 이야기가 달라진다. Byte Alignment 에 의해서 Structure 의 구조를 주고 받을 때는 영향을 받으며 Compiler 에 따라서 약간의 차이점 또한 생기게 된다. 이는 동일한 Language 를 사용하는 경우며, 만약 C++와 Ada Application 이 통신한다고 가정하면 그 문제는 더 더욱 복잡해 진다.



1-2-2. Conversion

System 의 수가 늘어감에 따라 위와 같은 1 대 1 의 Conversion 노력은 벽에 부딪치게 되었다. 위의 그림에서 보는 것 과 같이 시스템간의 네트워크가 발전하면서 연동이 필요한 급증함을 알 수 있다. 6 개의 장비가 있는 경우에 15 가지의 Conversion 이 개발 되어야 하며 각 시스템은 5 개의 Conversion 에 대한 노력을 기울여야 한다. N 시스템이 있는 경우 각 Vendor 는 N-1 의 Conversion 이 필요하게 된다. 이러 한계를 극복하기 위한 것이 ASN.1 의 등장이다. 각 시스템에서는 ASN.1 이라는 하나의 표준에 대한 Conversion 만을 개발하면 모든 장비와 Data 를 주고 받을 수 있게 된다. 이렇게 해서 만들어지게 된 것이 1988 년 ITU ASN.1 규격이다.

1-2-3. ASN.1 규격

최초의 ASN.1 표준 버전 X.409 가 1984 년 발표되었고, 이어서 새로운 버전이 1998 년 CCITT 의 X.208 과 1990 년 ISO 8824 가 발표되었습니다. 또한 ISO 8824 PDAM2(Prefixes and wild cards extensions) 의 표제 Part 1: Basic ASN.1 과 세가지 추가 부분: Information Object Specification, Constraint Specification, Parameterization 의 ASN.1 이 개정되었습니다.

1994 년에 아래의 표와 같이 ISO 와 CCITT 의 ASN.1 표준 문서가 발표되었고, 이 표준들은 2002 년에 개정되었습니다.

ITU-T X.680	ISO/IEC 8824-1
ITU-T X.681	ISO/IEC 8824-2
ITU-T X.682	ISO/IEC 8824-3
ITU-T X.683	ISO/IEC 8824-4
ITU-T X.690	ISO/IEC 8825-1
ITU-T X.691	ISO/IEC 8825-2

대부분의 abstract syntax 의 일반적인 사양은 아래와 같습니다.

- Modules and Assignments
  - Modules
  - Type Assignment
  - Value Assignment
- Built-in Types
  - Simple Types
  - Structured Types
- Tagged
- Useful Types
  - GeneralizedTime
  - UTCTime
  - EXTERNAL
  - Character String Types
  - Embedded PDV

- Additional Features
    - Subtype Notation and Value Sets
    - Recursion
    - Macros
    - Information Objects and Classes - *Coming Soon!*
    - Parameterized Types - *Coming Soon!*

(※ *Computer Networks and Open Systems An Application Development Perspective* by Lillian N. Cassel Richard H. Austing Jones & Bartlett Publisher)

※ ITU 참고사항 (2009 년 1 월) :

ITU-T ASN.1 Standards Projects

Abstract Syntax Notation One
X.680 – Basic Notation
X.681 – Information Objects
X.682 – General Constraint Notation
X.683 – Parameterization
X.690 – Basic (BER), Cannoical(CER), and Distinguished (DER) Encoding Rules
X.691 – Packed Encoding Rules (PER)

Latest
X.692 – Encoding Control Notation (ECN)
X.693 – XML Encoding Rules (XER)

For more information on ASN.1:
ITU-T ASN.1 Project (http://www.itu.int/itu-t/asn1)
Module Database (http://www.itu.int/itu-t/asn1/database/)
OID (Object Identifiers) (http://asn1.elibel.tm.fr/oid/)
ASN.1 Consortium (http://www.asn1.org)

ITU-T X.680-X.693 권고안

In force components		
Number	**Title**	**Status**
X.680 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation	In force
X.680-X.693 (07/02)	Information Technology - Abstract Syntax Notation One (ASN.1) & ASN.1 encoding rules This word collective file contains automatic cross-linkings, and is available in English only. Please read carefully the readme.rtf file before opening it.	In force
X.681 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Information object specification	In force
X.682 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification	In force
X.683 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications	In force
X.690 (07/02)	Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)	In force
X.691 (07/02)	Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)	In force
X.692 (03/02)	Information technology - ASN.1 encoding rules: Specification of Encoding Control Notation (ECN) An electronic version of Annex E of this Recommendation with an associated ECN Huffman encoding macro is also published independently and freely available from ITU website	In force
X.693 (12/01)	Information technology - ASN.1 encoding rules: XML encoding rules	In force

OID (Objective Identifiers) 권고안

X.660	Information technology - Open systems interconnection - Procedures for the operation of OSI registration authorities: General procedures and top arcs of the ASN.1 object identifier tree	PDF
X.662	Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Registration of values of RH-name-tree components for joint	PDF
X.667	Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components	PDF

Changing from ASN.1:1988 to ASN.1:2002(http://www.itu.int/ITU-

T/studygroups/com17/changing-ASN/)

규격	제목
X.208 (11/88)	**Specification of Abstract Syntax Notation One (ASN.1)**
X.209 (11/88)	**Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)**
X.660 (08/04)	Information technology - Open systems interconnection - Procedures for the operation of OSI registration authorities: General procedures and top arcs of the ASN.1 object identifier tree
X.667 (09/04)	Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components
X.680 (07/94)	**Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation**
X.680 (12/97)	Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation
X.680 (1997) Amendment 2 (06/99)	ASN.1 Semantic model
X.680 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation
X.680-X.693 (07/02)	Information Technology - Abstract Syntax Notation One (ASN.1) & ASN.1 encoding rules
X.681 (07/94)	**Information technology - Abstract Syntax Notation One (ASN.1): Information object specification**
X.681 (12/97)	Information technology - Abstract Syntax Notation One (ASN.1): Information object specification
X.681 (1997) Amendment 1 (06/99)	ASN.1 semantic model
X.681 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Information object specification
X.682 (07/94)	**Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification**
X.682 (12/97)	Information technology - Abstract Syntax Notation One (ASN.1): Constraint

	specification
X.682 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification
X.683 (07/94)	**Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications**
X.683 (12/97)	Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications
X.683 (1997) Amendment 1 (06/99)	ASN.1 semantic model
X.683 (07/02)	Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications
X.690 (07/94)	**Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)**
X.690 (12/97)	Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
X.690 (07/02)	Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
X.691 (04/95)	**Information technology - ASN.1 encoding rules - Specification of Packed Encoding Rules (PER)**
X.691 (12/97)	Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)
X.691 (07/02)	Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)
X.692 (03/02)	Information technology - ASN.1 encoding rules: Specification of Encoding Control Notation (ECN)
X.693 (12/01)	Information technology - ASN.1 encoding rules: XML encoding rules
X.694 (01/04)	Information technology - ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1
X.711 (1997) Technical Cor.2 (02/00)	Revision to include ASN.1:1997

X.722 (1992) Technical Cor.2 (02/00)	Revision of GDMO to include ASN.1:1997
X.723 (1993) Technical Cor.2 (02/00)	Revision of GMI to include ASN.1:1997
X.742 (1995) Technical Cor.2 (02/00)	Revision to include ASN.1:1997
X.744 (1996) Technical Cor.2 (02/00)	Revision to include ASN.1:1997
X.750 (1996) Technical Cor.1 (02/00)	Revision to include ASN.1:1997
X.751 (1995) Technical Cor.2 (02/00)	Revision to include ASN.1:1997
X.891 (05/05)	Information technology - Generic applications of ASN.1 - Fast INFOSET
X.892 (05/05)	Information technology - Generic applications of ASN.1 - Fast web services
Z.100 Supplement 1 (05/97)	SDL+ methodology: Use of MSC and SDL (with ASN.1)
Z.105 (03/95)	SDL combined with ASN.1 (SDL/ASN.1)
Z.105 (11/99)	SDL combined with ASN.1 modules (SDL/ASN.1)
Z.105 (10/01)	SDL combined with ASN.1 modules (SDL/ASN.1)
Z.105 (07/03)	SDL combined with ASN.1 modules (SDL/ASN.1)
Z.107 (11/99)	SDL with embedded ASN.1
Z.146 (03/06)	Testing and Test Control Notation version 3 (TTCN-3): Using ASN.1 with TTCN-3

※ ASN.1 & OID Project 관련 내용

The ASN.1 Project was established in February 2001 by ITU-T Study Group 7 to assist existing users of ASN.1 (ITU-T Rec. X.680, X.690 and X.890 series) within and outside of ITU-T, and to promote the use of ASN.1 across a wide range of industries and standards bodies. Since September 17, 2001, the responsibility for the ASN.1 Project resides with Study Group 17 and the Project now encompasses Object Identifiers (OIDs) and Registration Authorities (as defined in the ITU-T Rec. X.660 series).
(ITU-T Rec. X.680, X.690 and X.890 series)

Recommendation	Title	Format
X.680 (07/02) in force	Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation	PDF
X.680 Amendment 1 (10/03) in force	Support for EXTENDED-XER	PDF
X.680 Amendment 2 (08/04) in force	Alignment with changes made to ITU-T Rec. X.660 | ISO/IEC 9834-1 for identifiers in object identifier value notation	PDF
X.680 Corrigendum 1 in force		PDF
X.681 (07/02) in force	Information technology - Abstract Syntax Notation One (ASN.1): Information object specification	PDF
X.681 Amendment 1 (10/03) in force	Support for EXTENDED-XER	PDF
X.682 (07/02) in force	Information technology - Abstract Syntax Notation One (ASN.1): Constraint specification	PDF
X.683 (07/02) in force	Information technology - Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications	PDF
X.690 (07/02) in force	Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)	PDF
X.690 Amendment 1 (10/03) in force	Support for EXTENDED-XER	PDF
X.691 (07/02) in force	Information technology - ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)	PDF
X.691 Erratum 1 (06/03) in force	Erratum 1	PDF
X.691 Amendment 1 (10/03)	Support for EXTENDED-XER	PDF

in force		
X.691 Corrigendum 1 pre-published		PDF
X.692 (03/02) in force	Information technology - ASN.1 encoding rules - Specification of encoding control notation (ECN)	PDF
X.692 Amendment 1 (08/04) in force	Extensibility support	PDF
X.692 Annex E (03/02) in force	Support for Huffman encodings	PDF
X.692 Corrigendum 1 in force		PDF
X.693 (12/01) in force	Information technology - ASN.1 encoding rules: XML encoding rules (XER)	PDF
X.693 Amendment 1 (10/03) in force	XER encoding instructions and EXTENDED-XER	PDF
X.694 (01/04) in force	Information technology - ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1	PDF
Z.100 (08/02) in force	Specification and description language (SDL)	PDF
Z.100 Supplement 1 (05/97) in force	SDL+ methodology: Use of MSC and SDL (with ASN.1)	PDF
Z.100 Corrigendum 1 (08/04) in force		PDF
Z.100 Amendment 1 (10/03) in force	Backwards compatibility and compliance	PDF
Z.100 Annex F1 (11/00) in force	SDL formal definition: General	PDF
Z.100 Annex F2 (11/00)	Well-formedness and Transformation rules	PDF

in force		
Z.100 Annex F3 (11/00) in force	SDL formal definition: Dynamic Semantics	PDF
Z.110 (11/00) in force	Criteria for use of formal description techniques by ITU-T	PDF
Z.120 (04/04) pre-published	Message Sequence Charts (MSC)	PDF
Z.120 Annex B (04/98) in force	Annex B: Formal semantics of Message Sequence Charts	PDF
Z.140 (04/03) in force	Testing and test control notation version 3 (TTCN-3): Core language	PDF
Z.141 (02/03) in force	Testing and test control notation version 3 (TTCN-3): Tabular presentation format	PDF
Z.142 (02/03) in force	Testing and test control notation version 3 (TTCN-3): Graphical format	PDF

(ITU-T Rec. X.660 series)

X.660	Information technology - Open systems interconnection - Procedures for the operation of OSI registration authorities: General procedures and top arcs of the ASN.1 object identifier tree	PDF
X.662	Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Registration of values of RH-name-tree components for joint	PDF
X.667	Information technology - Open Systems Interconnection - Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components	PDF

1-2-4. ASN.1 1988 vs ASN.1 1997

TMN(Telecommunication Management Network)에서 사용되는 CMIP(Common Management Information Protocol)의 대부분의 규격은 ASN.1 1988 년을 사용하고 있다. 1988 년 버전은 초기 작품이라 Value 부분에 대한 부분은 parser 를 개발하기 거의 불가능하도록 되어 있는 등 많은 문제점을 안고 있다. ASN.1 1997 에서는 value Assignment 에 대한 문제점을 포함한 많은 문제들이 해결되어 1988 년에 비해 많이 사용 가능한 규격으로 생각된다. 1997 년 버전에서는 Open Type 의 개념이 소개 되는 대신 ANY type 을 삭제하였다. 하지만, 이전의 규격이 ANY type 을 많이 사용하기 때문에 1997 년 버전에 ANY 를 추가하여 사용하는 방법을 (비록 규격에는 위배되지만) Tool 개발 업체에서 채택하는 것을 볼 수 있다.

1-3. ASN.1 syntax 의 기본 형식

1-3-1 Abstract Syntax 와 Transfer Syntax

Abstract Syntax 와 Transfer Syntax 의 개념을 C 와 비교하면 이해하기 매우 쉽다. Abstract

Syntax 는 C 에서 Type 에 해당되는 것이다. Transfer Syntax 는 C 에는 개념이 없으며 그 이유는 간단하다. 예를 들어 int i 라고 선언된 변수의 값을 저장하는 것은 C Compiler 를 만드는 개발자에 자유게 맡긴다. 단지 i 에 대한 연산을 하거나 화면에 보여주는 경우에 올바르게 보여주면 되는 것이다. 하지만 ASN.1 의 경우에는 각 System 간의 통신을 해야 하기 때문에 int i 의 값이 어떤 식으로 encoding 이 되는지에 대한 규격이 필요하다. 이를 transfer syntax 라고 하여 integer 의 값은 어떻게 주고 받고, real 값은 어떻게 주고 받는 등의 규약이 정해져 있다. 처음으로 돌아가서 C 에서 굳이 Transfer Syntax 의 개념과 유사한 것을 찾는다면 int i 의 값이 저장되어 있는 형태라고 말할 수 있겠다.
Abstract Syntax 는 사용자를 위한 규격으로서 ITU 문서에 적혀있는 것은 모두 Abstract Syntax 를 사용한다. (X.208, X.209, X.680~X.683). 이 ASN.1 규격을 사용하여 실제 값을 주고 받을 때 Transfer Syntax 를 사용하게 되며 이에는 BER, PER, LWER(Light-Weight Encoding Rules) 등이 있다.

기본적인 ASN.1 의 기본 타입들은 tag number 와 클래스의(class) 종류, 조합형(constructed) 인가 단순형(primitive) 인가를 나타내는 플래그 등으로 구성된 헤더와 base contents 의 길이 그리고 base contents 등으로 구성됩니다.

CHOICE 와 ANY 타입을 제외한 모든 ASN.1 의 타입은 class 와 음이아닌 tag number 로 구성된 tag 를 가지고 있습니다. tag 의 클래스에는 Universal, Application, Private, Context-specific 등의 클래스가 있습니다.

Base content 의 길이는 직접 지정하는 definite 형식과 길이를 0 으로 설정하고 base contents 가 오고 마지막에 eoc(End of octet)를 나타내는 0000 으로 base contents 의 끝을 나타내는 indefinite 형식이 있습니다.

- Modules 과 Assignments

  A. module : ASN.1 의 기본 단위

  B. Type Assignment : 타입 할당

  예)

```
      InventoryList {1 2 0 0 6 1} DEFINITIONS ::=
        BEGIN
          {
           ItemId ::= SEQUENCE
             {
              partnumber IA5String,
              quantity INTEGER,
              wholesaleprice REAL,
              saleprice REAL
             }
           StoreLocation ::= ENUMERATED
             {
              Baltimore (0),
              Philadelphia (1),
              Washington (2)
            }
         }
       END
```

  C. Value Assignment : 값을 할당

  예)

```
  gadget  ItemId  ::=
   {
    partnumber      "7685B2",
    quantity        73,
    wholesaleprice  13.50,
    saleprice       24.95

    }
```

- Built-in Type

  A. Simple Types

Simple Types	Tag	Typical Use
BOOLEAN	1	Model logical, two-state variable values
INTEGER	2	Model integer variable values
BIT STRING	3	Model binary data of arbitrary length
OCTET STRING	4	Model binary data whose length is a multiple of eight
NULL	5	Indicate effective absence of a sequence element
OBJECT IDENTIFIER	6	Name information objects
REAL	9	Model real variable values
ENUMERATED	10	Model values of variables with at least three states
CHARACTER STRING	*	Models values that are strings of characters from a specified characterset

Character String Type	Tag	Character Set
NumericString	18	0,1,2,3,4,5,6,7,8,9, and space
PrintableString	19	Upper and lower case letters, digits, space, apostrophe, left/right parenthesis, plus sign, comma, hyphen, full stop, solidus, colon, equal sign, question mark
TeletexString (T61String)	20	The Teletex character set in CCITT's T61, space, and delete
VideotexString	21	The Videotex character set in CCITT's T.100 and T.101, space, and delete
VisibleString (ISO646String)	26	Printing character sets of international ASCII, and space
IA5String	22	International Alphabet 5 (International ASCII)
GraphicString	25	All registered G sets, and space
GraphicString	27	All registered C and G sets, space and delete

B. Structured Types

Structured Types	Tag	Typical Use
SEQUENCE	16	Models an ordered collection of variables of different type
SEQUENCE OF	16	Models an ordered collection of variables of the same type
SET	17	Model an unordered collection of variables of different types
SET OF	17	Model an unordered collection of variables of the same type
CHOICE	*	Specify a collection of distinct types from which to choose one type
SELECTION	*	Select a component type from a specified CHOICE type
ANY	*	Enable an application to specify the type **Note:** ANY is a deprecated ASN.1 Structured Type. It has been replaced with X.680 Open Type.

- Tagged

예) 아래의 a), b), c), d) 에서 tagging 이 없는 a) 는 잘 못된 방식의 문장입니다.

```
a)   seats  SET
        {
         maximum    INTEGER,
         occupied   INTEGER,
         vacant     INTEGER
        }

  b)   seats  SET
        {
         maximum   [APPLICATION 0] INTEGER,
         occupied  [APPLICATION 1] INTEGER,
         vacant    [APPLICATION 2] INTEGER
        }

  c)   seats  SET
        {
         maximum   [APPLICATION 0] IMPLICIT INTEGER,
         occupied  [APPLICATION 1] IMPLICIT INTEGER,
         vacant    [APPLICATION 2] IMPLICIT INTEGER
        }

  d)   seats  SET
        {
```

```
        maximum   [0] INTEGER,
        occupied  [1] INTEGER,
        vacant    [2] INTEGER
       }
```

- Useful Types

A. GeneralizedTime

B. UTCTime

C. EXTERNAL

예)

ISO 8824 는 아래의 EXTERNAL type 을 정의하고 있다.:

```
EXTERNAL  ::=  [UNIVERSAL 8] IMPLICIT SEQUENCE
{
 direct-reference  OBJECT IDENTIFIER OPTIONAL,
 indirect-reference  INTEGER OPTIONAL,
 data-value-descriptor  ObjectDescriptor  OPTIONAL,
 encoding  CHOICE
             {single-ASN1-type  [0] ANY,
              octet-aligned     [1] IMPLICIT OCTET STRING,
              arbitrary         [2] IMPLICIT BIT STRING}
}
```

D. Character String Types

Character Type	Tag	Description
BMPString	30	Basic Multilingual Plane of ISO/IEC/ITU 10646-1
IA5String	22	International ASCII characters (International Alphabet 5)
GeneralString	27	all registered graphic and character sets plus SPACE and DELETE
GraphicString	25	all registered G sets and SPACE
NumericString	18	1, 2, 3, 4, 5, 6, 7, 8, 9, 0, and SPACE
PrintableString	19	a-z, A-Z, ' () +,-.?:/= and SPACE
TeletexString	20	CCITT and T.101 character sets
UniversalString	28	ISO10646 character set
UTF8String	12	any character from a recognized alphabet (including ASCII control characters)
VideotexString	21	CCITT's T.100 and T.101 character sets
VisibleString	26	International ASCII printing character sets

E. Embedded PDV

예)

```
  EmbeddedPDV ::= [UNIVERSAL 11] IMPLICIT SEQUENCE {
  identification CHOICE {
    syntaxes SEQUENCE {
      abstract OBJECT IDENTIFIER,
      transfer OBJECT IDENTIFIER }
      -- Abstract and transfer syntax object identifiers --,
      syntax OBJECT IDENTIFIER
      -- A single object identifier for identification of the abstract
      -- and transfer syntaxes --,
      presentation-context-id INTEGER
      -- (Applicable only to OSI environments)
      -- The negotiated OSI presentation context identifies the
      -- abstract and transfer syntaxes --,
      context-negotiation SEQUENCE {
              presentation-context-id INTEGER,
              transfer-syntax OBJECT IDENTIFIER }
      --(Applicable only to OSI environments)
      -- Context-negotiation in progress, presentation-context-id
      -- identifies only the abstract syntax
      -- so the transfer syntax shall be specified --,
      transfer-syntax OBJECT IDENTIFIER
      -- The type of the value (for example, specification that it is
      -- the value of an ASN.1 type)
      -- is fixed by the application designer (and hence known to both
      -- sender and receiver). This
      -- case is provided primarily to support
      -- selective-field-encryption (or other encoding
      -- transformations) of an ASN.1     type --,
      fixed NULL
      -- The data value is the value of a fixed ASN.1 type (and hence
      -- known to both sender
      -- and receiver) --
    },
    data-value-descriptor ObjectDescriptor OPTIONAL
    -- This provides human-readable identification of the class of the
    -- value --,
    data-value OCTET STRING }
  ( WITH COMPONENTS {
         ... ,
              data-value-descriptor ABSENT } )
```

- Additional Features

A. Subtype Notation and Value Sets

B. Recursion

C. Macros

D. Information Objects and Classes

E. Parameterized Types

- Listing of Universal Tags

Universal Tag Number	Description
0	reserved for BER
1	BOOLEAN
2	INTEGER
3	BIT STRING
4	OCTET STRING
5	NULL
6	OBJECT IDENTIFIER
7	ObjectDescriptor
8	INSTANCE OF, EXTERNAL
9	REAL
10	ENUMERATED
11	EMBEDDED PDV
12	UTF8String
13	RELATIVE-OID
16	SEQUENCE, SEQUENCE OF
17	SET, SET OF
18	NumericString
19	PrintableString
20	TeletexString, T61String
21	VideotexString
22	IA5String
23	UTCTime
24	GeneralizedTime

25	GraphicString
26	VisibleString, ISO646String
27	GeneralString
28	UniversalString
29	CHARACTER STRING
30	BMPString

- 예제(Exercises)

1. Given the definition

```
company  ::=  SET
             {
              name           [0]  IA5String,
              zipcode        [1]  IA5String,
              CitationType        INTEGER,
              other               ANY DEFINED BY CitationType
              }
```

where the INTEGER value of CitationType can be 0 = INTEGER, 1 = REAL, or 2 = BOOLEAN, which of the following values are valid? Assume tagging has been done accurately.

   1. ``CyberReal", ``20742-1911", 1, TRUE
   2. ``60603", ``Villaland", 0, 500000.00
   3. 0, 450, ``HomeNet"
   4. ``SitCom", 1, 70000.00

2. Write a module that identifies DaysOfWeek as a BIT STRING consisting of seven bits, one for each day of the week and the first of which represents Sunday. Write a value of the string that represents (Monday, Wednesday, Saturday).

3. Differentiate between the following two representations?

```
a.  HouseType  ::=  INTEGER       b.  HouseType  ::=  ENUMERATED
      (                                 {
       Ranch        (1)                  Ranch        (1)
       SplitLevel   (2)                  SplitLevel   (2)
       Colonial     (3)                  Colonial     (3)
       TownHome     (4)                  TownHome     (4)
       }                                  }
```

4. Assume Employee is of type SEQUENCE with the following components: hire date, job title, age, salary, and office location (a city name). Write an ASN.1 notation for Employee. Give a valid representation of Employee.

5. Distinguish among the various kinds of tagging in ASN.1. Use an example to illustrate the distinctions. Indicate the effects of IMPLICIT and EXPLICIT.

6. Differentiate among SEQUENCE, SET, ENUMERATED, and WITH COMPONENTS. In your discussion, use the following structures as examples to illustrate differences.

```
a) airport ::= SEQUENCE
                 {
                   origin       [0]  IA5String,
                   stop         [1]  IA5String  OPTIONAL,
                   destination  [2]  IA5String
                 }

b) airport ::= SET
                 {
                   origin       [0]  IA5String,
                   stop         [1]  IA5String,
                   destination  [2]  IA5String
                 }

c) airport ::= ENUMERATED
                 {
                   origin       [o],
                   stop         [s],
                   destination  [d]
                 }

d) airport_list ::= airport
               ( WITH COMPONENTS
                 {
                   origin       [0],
                   stop         [1],
                   destination  [2]
                 }
           )
```

7. Give an instance of the AirlineFlight example in Structured Types Section that includes a stop in the Dallas-Fort Worth (DFW) airport.

8. For each of the following examples, name an appropriate ASN.1 data type and write the corresponding ASN.1 definition.

    1. An alphabetized list of employees.
    2. One of a movie, play, or sport event.
    3. Prime numbers between 0 and 15.
    4. 110010001111000011111100
    5. The local time in hours, minutes, and seconds.
    6. A sentence of text.
    7. Number of cars delivered, sold, leased, and on-hand.

9. The following data structures are written in C. Write each structure in ASN.1.

```
a) struct calendar birthday[2] =
     { { {`O', `C', `T'}, 2, 1948 }, { {`A', `P', `R'}, 14, 1955 } }

   where "calendar" is a structure defined by

   struct  calendar
     {
      char  name[3];
      int   date:
      int   year;
     };

b) struct time  depart_time, arrive_time;

   where "time" is a structure defined by

     struct  time
       {
        int  hour;
        int  minute;
        int  second;
       };

c) char array[7] = "NETWORK"

d) struct entry
     {
      char  *word;
      int   *page_number;
     } index[50] =
     {  {"ARPANET", 105}, {"ASN.1", 328}  };
```

10. Write in Pascal (or another high-level language different from C) each of the ASN.1 representations you produced in the preceding exercise.

11. CASE: Stores in Philadelphia and Washington, but not Baltimore, carry colonial flags. The stores obtain the flags for 25.99 each. The identification number for colonial flags is cf1783, The Philadelphia store has 14 flags on hand and the Washington store has 10. Write the ASN.1 representation(s) for this CASE using the module in this Figure.

12. Write a definition of NonStopFlights in this Section with the following additional conditions: include United Airlines and allow both nonstop flights and those that stop in Dallas-Fort Worth (DFW).

13. Consider the entries in the Table of Contents for this chapter as records in ``ASN.1".

    1. Which main section headings represent simple records and which represent structured records?

    2. Write an ASN.1 recursive definition that specifies a record such as ``ASN.1".

    3. Write the Table of Contents as an instance of the record definition.

14. Write an ASN.1 recursive definition of a binary tree.

15. Use your definition of a binary tree in the preceding exercise to represent the following instance:

```
              50
             /  ₩
           /      ₩
        42          60
       /  ₩        /
    30     47   58
      ₩
        25
```

ASN.1 기본 문장을 살펴보면 아래와 같다.

TypeReference ::= TagInfo Type

예제)

Bool ::= BOOLEAN

X ::= [PRIVATE 12] INTEGER

Y ::= [APPLICATION 22] EXPLICIT Bool

Z ::= SEQUENCE { a X, b Y }

1. TypeReference 는 식별자이고, 이름이다.
2. 문장은 (A..Z) 대문자로 시작하며, 어떠한 숫자와 문자의 조합을 포함 할 수 있다.
3. 특수 문자 하픈(hyphen “ -“) 은 한 개만 허용하며, 두 개의 하픈(“ —“)은 주석(comment) 이 된다.

TagInfo 은 메시지(message) 속에 인코드된(encoded) Tag 를 정의한다.

Syntax is [<Class> <Identifier>]

<Class> : 00 - UNIVERSAL, 01 - APPLICATION, 10 - CONTEXT, 11 - PRIVATE

<Identifier> : 인코드된 식별자 번호 (Encoded identifier number)

TagInfo 은 옵션 사항이고, 디폴트 tag info 가 사용될 수 있다.

Type 은 아래 중의 하나의 ASN.1 Type 을 정의한다.

- 내장된 단순형 Built-in Primitive (예, BOOLEAN, INTEGER, etc.)
- 내장된 조합형 Built-in Constructed (예, SEQUENCE, SET, CHOICE, etc.)
- 정의형 Defined : 앞서 선언된 Type 을 위한 참조

ASN.1 의 기본적인 Syntax 의 C/C++ Mapping 은 아래와 같다.

ASN1C:	Name ::= Type
Maps to(+>)	typedef <C-Type> Name;

ASN1CPP:	Name ::= Type
Maps to (=>)	typedef <C-Type> AS1T_Name;
	class ASN1C_Name;

ASN.1 기본 변수 Notation Syntax 는 아래와 같다.

ValueReference Type ::= Value

ValueReference 는 변수에 할당된 식별자이다.

C 언어의 변수 이름과 유사하다. (예. Int C)

반드시 소문자(a..z)로 시작하고, 그 다음은 문자와 숫자가 포함될 수 있다.

특수문자로는 한 개의 하픈 " -" 허용한다.

ASN.1 Type 에 정의된 Type 은 내장된(Built-in) 또는 정의된(Defined) 형 이다.

Value syntax 는 그 type 에 의존한다.

예.) Max-limit INTEGER ::= 10000

내장된 데이터의 타입 (Built-in Data Type)

1. 대부분 implicit, universal 태그로 구성된다.(CHOICE 제외)

2. PRIMITIVE(단순형) 와 CONSTRUCTED(복합형) 의 두 개의 타입이 있다.

3. PRIMITIVE 는 BOOLEAN, INTEGER, ENUMERATED, NULL, 등이 있고, 데이터 content 의 정렬을 직접 encode 한다.

4. CONSTRUCTED 타입은 한 개를 포함하고 있거나 또는 그 보다 많은 PRIMITIVE 또는 CONSTRUCTED encode 를 포함하고 있다. 이것 들은 " containers" 또는 " wrappers" 처럼 취급 될 수 있다.

단순형 데이터의 타입 (Primitive Built-in Data Types): BOOLEAN 예제

1. Syntax: <Name> ::= BOOLEAN
2. Maps to " typedef ASN1BOOL <Name>;" (ASN1BOOL 은 C 의 unsigned char type 이다.)
3. Value notation: TRUE or FALSE (참 또는 거짓으로 표기 된다.)

Simple Types	Tag	Typical Use
BOOLEAN	1	Model logical, two-state variable values
INTEGER	2	Model integer variable values
BIT STRING	3	Model binary data of arbitrary length
OCTET STRING	4	Model binary data whose length is a multiple of eight
NULL	5	Indicate effective absence of a sequence element
OBJECT IDENTIFIER	6	Name information objects
REAL	9	Model real variable values
ENUMERATED	10	Model values of variables with at least three states
CHARACTER STRING	*	Models values that are strings of characters from a specified characterset

Character String Type	Tag	Character Set
NumericString	18	0,1,2,3,4,5,6,7,8,9, and space
PrintableString	19	Upper and lower case letters, digits, space, apostrophe, left/right parenthesis, plus sign, comma, hyphen, full stop, solidus, colon, equal sign, question mark
TeletexString (T61String)	20	The Teletex character set in CCITT's T61, space, and delete
VideotexString	21	The Videotex character set in CCITT's T.100 and T.101, space, and delete

VisibleString (ISO646String)	26	Printing character sets of international ASCII, and space
IA5String	22	International Alphabet 5 (International ASCII)
GraphicString	25	All registered G sets, and space
GraphicString	27	All registered C and G sets, space and delete

조합형 데이터의 타입 (Constructed Built-in Data Types)

Structured Types	Tag	Typical Use
SEQUENCE	16	Models an ordered collection of variables of different type
SEQUENCE OF	16	Models an ordered collection of variables of the same type
SET	17	Model an unordered collection of variables of different types
SET OF	17	Model an unordered collection of variables of the same type
CHOICE	*	Specify a collection of distinct types from which to choose one type
SELECTION	*	Select a component type from a specified CHOICE type
ANY	*	Enable an application to specify the type **Note:** ANY is a deprecated ASN.1 Structured Type. It has been replaced with X.680 Open Type.

SEQUENCE, SET, SEQUENCE OF, SET OF, 그리고 CHOICE 의 타입이 있다.

“ Hole” 타입으로도 부른다. (ANY, ANY DEFINED BY, EXTERNAL, 기타.)

선 정의된 타입 (Defined Types)

먼저 선언된 Type 을 참조 한다. 예제;

```
X ::= INTEGER

Y ::= X
```

생성된 코드는 먼저 선언된 타입을 참조 한다.;

```
typedef ASN1INT X;

typedef X Y;
```

ASN.1 은 일련의 형 식별자(identifier)를 제공하며, 다음의 네 가지 클래스로 구분된다.

00. UNIVERSAL 정수와 같은 일반형

01. APPLICATION 전체 응용 개체에 공통인 형

10. CONTEXT 사용하는 특정 문맥과 관련된 형

11. PRIVATE 사용자 정의형

ASN.1 에서 사용하는 CONSTRUCTED(복합형)

1. UNIVERSAL (constructed) SEQUENCE :

   고정되고 순서가 있는 형들의 리스트. 일부는 선택적으로 선언될 수 있다. 즉 관련된 형의 값이 순서를 구성하는 개체(entity) 에 의해 배제될 수 있다.

2. SEQUENCE OF : 모든 원소가 동일한 형을 가진 고정되거나 비경계이며 순서가 있는 배열

3. SET : 몇 개의 원소가 선택적으로 선언 될 수 있는 고정되고 순서가 없는 형의 리스트

4. SET OF : 모든 원소가 동일한 형을 갖는 고정되거나 비경계이며, 순서가 없는 원소의 배열

5. CHOICE : 미리 정의된 형의 세트에서 선택된 고정적이며 순서가 없는 형의 리스트

1-3-2 Sequence 와 Set 의 차이점

ASN.1 에서 혼란을 시키는 것이 있는데 그 중의 하나가 SEQUENCE 와 SET 이다.

SEQUENCE 는 C 의 struct 와 동일한 개념이다.
SET 의 개념은 C 에는 없고, 굳이 찾는 다면 C 의 struct 에서 순서가 무의미한 type 으로 보면 된다.

```
Seq ::= SEQUENCE {
first INTEGER,
second INTEGER
}

Set ::= SET {
first INTEGER,
second INTEGER
}

seqVa1 Seq ::= { 10, 20 }
seqVal2 Seq ::= { 20, 10 }

setVal1 Set ::= { 10, 20 }
setVal2 Set ::= { 20, 10 }
```

seqVal1 과 seqVal2 의 경우에는 서로 다른 값이지만 setVal1 과 setVal2 는 동일한 값이다.

1-3-3 Sequence 와 Sequence Of 의 차이점

이 둘은 이름만 비슷하지 전혀 다는 개념이다. SEQUENCE 는 C 의 struct 이고 SEQUENCE OF 는 C 의 Array 선언 [ ] 에 해당된다.
Name ::= SEQUENCE OF INTEGER 는 int Name[ ] 과 동일한 의미이다.
SET 과 SET OF 도 유사하다.

1-3-4 Sequence Of 와 Set Of 의 차이점

SEQUENCE 와 SET 과 차이처럼 Array 내 값의 순서가 의미를 가지느냐 여부의 차이이다.

```
SeqOf ::= SEQUENCE OF INTEGER
SetOf ::= SET OF INTEGER

seqOfVal1 SeqOf ::= { 10, 20, 30 }
seqOfVal2 SeqOf ::= { 20, 30, 10 }
setOfVal1 SeqOf ::= { 10, 20, 30 }
setOfVal2 SeqOf ::= { 20, 30, 10 }
```

seqOfVal1 와 seqOfVal2 는 다른 값이지만 setOfVal1 와 setOfVal2 는 동일한 값이다.

1-3-5 Choice 개념

CHOICE 는 C 의 union 과 동일한 개념이다. CHOICE 에 값을 선언할 때 선언되어 있는 Element 중 하나를 사용하면 된다.

```
Choice ::= CHOICE {
int INTEGER,
string PrintableString
}

chocieVal1 Choice ::= int : 10
choiceVal2 Choice ::= string : " asn.1 is fun"
```

위의 예제는 1997 년 ASN.1 표준을 따른 것이며 : 앞의 int 와 string 은 Choice 의 Alternative 의 identifier 를 가리킨다.

1-3-6 TLV Encoding (BER)

송수신의 사용되는 Transfer Syntax 은 TLV 형식을 취한다. T 는 Type(Identified by tagging)이고, L 은 Length 그리고 V 는 Value 이다.

TYPE(Identified by tagging)	LENGTH	VALUE

Type 은 Class, Form 그리고 Identifier code 의 3 가지 요소를 가지고 있다.

8 7	6	5 4 3 2 1
C(Class)	F(Form)	ID(Identifier)

Class 의 비트(bit) 는 아래와 같다.

00	UNIVERSAL : 표준 그룹에 의한 표준 태그가 할당되어 있다.
01	APPLICTION : 할당된 어플리케이션을 위한 태그들이 정의 되어있다.
10	CONTEXT : 일시적인 태그로 할당된 context 내에서 다시 사용될 수 있다.
11	PRIVATE : 사용자을 위한 태그이다

Form 의 비트(bit) 는 아래와 같다

0	PRIMITIVE: (단순형) Primitive encoding (value contains data)
1	CONSTRUCTED : (조합형) Constructed encoding (value contains 1 or more TLV's)

ID (Identifier code) 는 인코드된 식별자의 번호다.

Integer Type 의 예를 들어보면 간단하다.
i ::= INTEGER 10

T	L	V
INTEGER	V 의 길이	10

실제 전송되는 값은 T L V 에 해당된다.

T 는 INTEGER 에 해당하는 값으로 채워지게 된다. Value 는 Integer 10 을 나타내는 값이 들어가고 L 인 Length 는 INTEGER i 의 길이에 대한 정보를 담고 있게 된다. 주의해야 할 것은 i 라는 정보는 전송되지 않는다는 점이다. 변수 이름인 i 는 사용자를 위한 것이다.

+Boolean 이나 Enumerated 와 같이 Simple 한 Type 의 경우에는 T L V 중 T 의 값이 변하게 되지만, Sequence, Set, Choice 와 같은 경우에는 다른 Simple Type 을 포함하는 형태로 전송되어 진다.

```
Seq ::= SEQUENCE {
first INTEGER,
second INTEGER
}
seqVa1 Seq ::= { 10, 20 }
```

Seq 의 경우에 T L V 의 V 부분에 SEQUENCE 의 element 에 해당되는 INTEGER 인 first 와 second 의 내용이 TLV 형태로 들어가게 된다. 이를 그림으로 나타내면 아래와 같다.

T	L	V		
SEQUENCE	V 의 길이*	T	L	V
		INTEGER	V 의 길이	10
		INTEGER	V 의 길이	20

* SEQUENCE 의 L 부분은 element 인 first 와 second 의 길이 모두를 나타낸다.

TLV Encoding 에서 제일 특이한 Type 이 Choice 인데 Choice 에 해당하는 Type 은 없다. Choice 는 alternative 중에서 하나를 사용하기 때문에, 실제로 전송되는 값에는 선택된 alternative 의 Type 만으로 T L V 의 형태를 띄게 된다. Sequence 의 Embedded Format 과는 사뭇 다르게 된다.
NULL Type 이나 SEQUENCE OF, SET OF 그리고 Optional 을 사용하는 경우, Length 가 0 인 경우도 발생한다는 사실이다. 결과적으로 T L(0) 만 나타나게 된다.

1-3-7 Tag 의 개념

TAG 은 Data 를 상호간에 전송 시 Encoded data 를 수신할 때 나타나는 문제점을 해결하기 위함이다. 간단한 예를 들어보면 쉽게 이해할 수 있다. 아래와 같은 SEQUENCE 의 경우를 보자.

```
Seq ::= SEQUENCE {
one INTEGER OPTIONAL,
two INTEGER OPTIONAL
}
seqVal ::= Seq { one 10 }
```

위의 값이 Encoding 되면 T(SEQ) L T(INTEGER) L V(10)으로 보내어진다. 허나 이를 수신하는 측에서는 T(INTEGER)가 one 인지 two 인지 알 수 있는 방법이 없게 된다. 이를 위해서 소개된 개념이 Tag 이다. 이와 같이 수신측에서 decoding 할 수 없는 규격의 경우에는 아래와 같이 Tag 정보를 추가해 사용한다.

```
Seq ::= SEQUENCE {
one [1] INTEGER OPTIONAL,
two [2] INTEGER OPTIONAL
}
seqVal ::= Seq { one 10 }
```

위의 경우에는 T(SEQ) L T([1]) L V(10)처럼 3 번째의 Tag 정보가 INTEGER 대신 [1]의

정보가 보내어진다. 수신측에서는 [1]이 one 을 의미하는 것을 알기 때문에 V(10)이 integer type 이라는 것을 알 수 있으며, 결과적으로 decoding 에 전혀 문제가 없게 된다. 이와 유사하게 Set 과 Choice 의 경우에도 Tag 가 사용된다.

1-3-8 IMPLICIT TAG, EXPLICIT TAG, AUTOMATIC TAG

```
Seq ::= SEQUENCE {
one [1] INTEGER OPTIONAL,
two [2] INTEGER OPTIONAL
}
seqVal ::= Seq { one 10 }
```

Tag 설명 부분에서 위의 seqVal 이 T(SEQ) L T([1]) L V(10)으로 encoding 되어 진다고 하였는데 이는 IMPLICIT TAG 인 경우에 해당한다. Implicit 이 의미하는 것은 [1]의 Tag 정보만으로 실제의 type 을 알 수 있기 때문에 굳이 실제의 Type 을 사용하지 않는다는 것으로 의미한다.

이와는 달리 Tag 의 정보를 사용하고 실제 Type 의 정보 또한 보내도록 하는 Option 을 사용할 수 있다. 이를 EXPLICIT TAG 이라고 하고 이 경우에는 encoding 이 아래와 같이 되어진다.

T(SEQ) L T([1]) L T(INTEGER) V(10)
예상한대로 IMPLICIT TAG 와 EXPLICIT TAG 을 사용하는 규격간에는 동일한 값이라도 encoding 이 다르게 된다는 것을 인식해야 한다.

AUTOMATIC TAG 은 1997 년에 소개된 개념으로써 위와 같은 경우에 일일이 수작업으로 규격에 TAG 을 붙이는 것이 귀찮기 때문에 자동적으로 Tag 을 붙이는 기능이다. ITU 에서는 이 기능의 사용을 적극적으로 권하고 있지만 TMN 규격들이 이전에 선언되었기 때문에 사용되는 일은 거의 없다.

1-3-9 동일한 A S N . 1 규격의 사용

ASN.1 규격은 두개 이상의 Node 가 정보를 주고 받기 위하여 만들어 진 것이다. ASN.1 규격이란 이 정보가 어떤 형태를 가지는 것을 적어 놓은 것이다. 따라서 ASN.1 규격은 통신을 하는 모든 Node 가 동일한 혹은 Compatible 한 ASN.1 규격을 가지고 있어야 한다.

1-3-10 ASN.1 규격의 Compatibilty

ASN.1 규격을 직접 만들거나, 정해진 ASN.1 규격을 구현을 위해 사용하는 경우에, 사용하는 Tool 의 제한 사항 등으로 인해 나름대로의 수정이 필요하게 된다. 이때 고민하게 되는 것이 수정된 ASN.1 규격이 과연 기존의 ASN.1 규격과 통신 시 문제를 발생시키지 않을까 하는 걱정이 생기게 된다.

이름은 사용자를 위한 것
ASN.1 규격상 모든 이름은 사용자를 위한 것이다. 예를 들어, i ::= INTEGER 10 에서 i 란 이름은 사용자를 위한 것이기 때문에 해당규격에서 모든 i 를 newname 으로 바꾸어도 상관없다.

D e f i n e d T y p e 과 직접 사용
아래의 예제와 같이 INTEGER 를 직접 사용하던지, Integer 란 Type 을 define 하여 사용하던지

이는 동일하다. Element 가 Sequence 나 Set 의 경우에도 동일하게 적용된다.

```
Seq ::= SEQUENCE {
   one INTEGER,
   two INTEGER
}
Integer ::= INTEGER
Seq ::= SEQUENCE {
   one Integer,
   two INTEGER
}
```

사용하지 않는 규격의 가지치기
Workstation 이나 Windows 의 환경에서는 메모리에 대한 제한이 거의 없지만 Embedded 환경에서 ASN.1 관련 작업을 하는 경우에는 메모리에 대한 부담을 가지게 된다. 이런 경우에는 ASN.1 규격상에서 절대로 사용하지 않는 부분을 제게 하는 것을 고려해 봄
직하다. 하지만, ASN.1 규격이라는 것이 모두 연결되어 있기 때문에 제거하는 것이 쉽지는 않다.

```
Select ::= CHOICE {
   this SomeSequence,
   that SomeSequence,
n e v e r S e q u e n c e N e v e r U s e d
}
```

never 가 구현하는 application 에서 사용하지 않는다는 것을 확신할 수 있으면 SequenceNeverUsed 의 type 선언과 여기서 파생되는 사용하지 않는 선언을 모두 삭제할 수 있다. 사용하지 않더라고 그대로 사용하면 parsing 에 필요한 code 들이 생성되기 때문에 그 만큼의 ROM 을 필요로 하기 때문이다.
만약, 실제 상황에서 제거된 element 가 수신되는 경우에는 어떤 일이 발생할까?
간단하다. 수신 측에서 분석이 불가능하다는 Parsing error 가 발생하게 된다.

동일한 D e f i n e d T y p e 의 통합
아래와 같이 동일한 definition 이나 다른 이름을 사용하는 경우가 간혹 있다. 이는 가지치기를 한 후에 더 많이 생겨날 수 있는 가능성이 있는데 이 경우에 하나의 type 으로 통일하여주면 formating 과 parsing 에 필요한 code 들을 줄일 수 있다.

```
Select ::= CHOICE {
   this ThisSequence,
   that ThatSequence
}
ThisSequence ::= SEQUENCE {
   one INTEGER,
   two INTEGER
}

ThatSequence ::= SEQUENCE {
   one INTEGER,
   two INTEGER
}
```

위의 Choice 는 아래와 같이 바꿀 수 있다.

```
Select ::= CHOICE {
   this ThisSequence,
   that ThisSequence
}
```

ASN.1 Compiler

통신을 하기 위해서는 주어진 규격에 맞추어 값을 채우고 이를 ASN.1 Transfer Syntax 에 맞는 형태의 byte stream 을 만들어 주어야 한다. 그래야 PDU 형태로 송신을 할 수 있기 때문이다. ASN.1 Compiler 는 사용자의 ASN.1 규격을 읽어드려, 이에 대한 Validity 를 확인하고, header file 과 두 가지 종류의 c file 을 만들어 낸다 – Formatter & Parser. Header file 은 사용자가 해당 type 을 선언하여 사용할 때 사용하고 Formatter 는 사용자의 값을 bit stream 으로 바꾸어 주는 역할을 한다. Parser 는 이와 반대로 bit stream 을 header file 의 structure 값으로 변환 시켜주는 기능을 수행한다.

이해를 위해 예를 들어 본다. 이는 특정 ASN.1 Compiler 에 해당되는 내용이 아니며, ASN.1 Compiler 마다 처리 방법은 각각 다르게 나타난다.

```
Seq ::= SEQUENCE {
   first INTEGER,
   second INTEGER
}
seqVa1 Seq ::= { 10, 20 }
```

[Header File]
"asn1header.h"

```
struct Seq {
   int first;
   int second;
}
```

사용자는 위의 header 을 읽어드려서 송신을 위한 아래의 ASN.1 값을 만들게 된다.
[사용자 Code]

```
#include "asn1header.h"
struct Seq asn1value = { 10, 20 };
```

이를 송신하기 위해서 ASN.1 Value 를 bit stream 으로 바꾸게 된다.

생성된 c code 를 살펴보면 ASN.1 의 각 type 마다 formatter/parser 가 생성됨을 알 수 있다. 사용되는 interface 는 아래와 같다.

```
/* returns length */
int format_Seq(struct Seq *value; char *bitStream);
```

사용자는 준비된 asn1value 를 formatter 에 넣어주면 bitStream 에 ASN.1 encoded 가 생성되고 그 길이가 return 되어진다. Encoding 되어진 값은 LAN 과 같은 통신 채널을 통해 상대편 Node 에 전해진다.

ASN.1 Encoded 값을 수신 받으면 아래의 parser 를 이용하여 program 에서 사용 가능한 structure 의 형태로 바꾸게 된다.

Parser_Seq(char *bitStream, int len, struct Seq *value);

실제의 ASN.1 header, formatter, parser 는 당연히 이보다 훨씬 더 복잡하게 되어 있다.
위의 예제는 ASN.1 Compiler 에서 사용되는 개념만을 전달하기 위한 것으로, 이해를 위해서 최대한 단순화 시킨 것이다.

Object Identifier
Object Identifier 는 이름 그대로 OSI 에서 사용되고 있는 Object 를 나타내기 위한 ID 이다.
Object Identifier 는 { 2 9 3 5 2 0 }과 같이 생겼으며 아래와 같이 이름이나 이름과 숫자를 혼용하여 사용되어 진다. { joint-itu-ccitt ms(9) smi(3) asn1Module(2) attributes(0) }
ASN.1 에는 OBJECT IDENTIFIER type 이 지원된다.

그외 설명되지 않은 것들
ANY Type, EXTERNAL Type, SubType, Macro, 그리고 Object, Parameter 개념 등을 포함하여 많은 내용이 설명되어 있지 않다. 위의 개념을 포함한 모든 ASN.1 개념을 이해하는 것은 상당한 노력이 필요하다.

ASN.1 Definitions
ASN.1 규격을 가지고 작업하다 보면 IMPORT 된 type reference 나 value reference 를 찾는데 애를 먹게 되는 경우가 있다. 만약 ITU 규격에 선언되어 있는 것이라면 ITU ASN.1 Module Database 를 사용하면 쉽게 찾을 수 있다.

또한 ASN.1 Definition 이 필요한 경우 ITU 규격에서 드래그하여 사용해야 하는데 불편하다.
가끔씩 오류도 있고. 일반적으로 많이 사용되는 ASN.1 definition 의 경우에는 Tool Vendor 에서 제공되나, 부족한 경우에는 이곳에서 필요한 ASN.1 Definition 을 가져다 사용하면 수정 없이 그대로 사용 가능하다.

http://www.itu.int/ITU-T/asn1/database/index.html
http://www.itu.int/search/index.html 사용

1-4. BER 과 DER

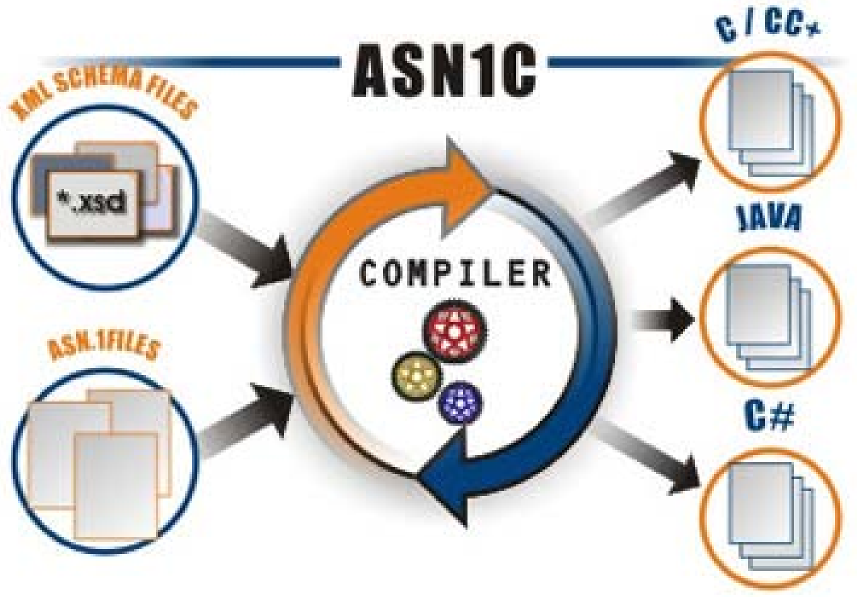
BER 은 ASN.1 의 기본 인코딩 규칙**(Basic Encoding Rules)**을 나타내는 일반적인 이름입니다. BER 는 ITU-T 권고안 X.209 및 X.690 에 정의되어 있습니다. BER 는 통신 링크를 통해 전송할 수 있는 옥텟의 스트림으로 ASN.1 데이터를 인코딩하는 하나의 규칙 집합입니다. ASN.1 데이터를 인코딩하는 다른 방법으로는 **DER(Distinguished Encoding Rules), CER(Canonical Encoding Rules)** 및 **PER(Packed Encoding Rules)**가 있습니다.

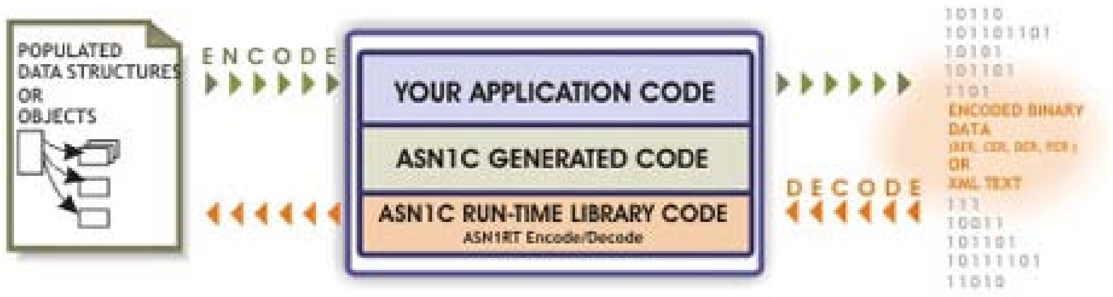
네트워크 상에서 교환되는 메시지는 ASN.1 형태로 구성되어 교환된다고 하였지만, ASN.1 은 그 자체로는 추상적인 데이터 형식이기 때문에 그것을 그대로 전송할 수 없습니다. 따라서 ASN.1 형태의 추상 데이터 형식을 네트워크상에서 전송할 수 있는 형식으로 변환시켜야 할 필요성이 있습니다. 이때 사용되는 것이 BER(Basic Encoding Rule)과 DER(Distinguished Encoding Rule)이다. BER 과 DER 은 각 ASN.1 형태의 데이터 형식을 네트워크 상에서 전송할 수 있는 형태로 encoding 해주고 그것을 다시 ASN.1 형태로 decoding 해주는 방법에 대하여 정의하고 있습니다.

대부분의 ASN.1 형식에 대해서 BER 과 DER 방식의 encoding 방법은 큰 차이를 보이지 않습니다. 다만 경우에 따라서 약간의 다른점들이 존재하며, BER 이 DER 보다 더 큰 범위의 encoding 방법을 정의하고 있습니다.

1-5. ASN1C & ASN1CPP

ASN1C 은 ASN.1 의 C 소스 코드 컴파일러입니다. 이 컴파일러는 ASN.1 의 소스 파일을 입력으로 취하여 C typedefs 를 포함하는 헤더 파일과 encode 와 decode 함수들을 포함하는 C 소스 파일을 생성하여 줍니다.

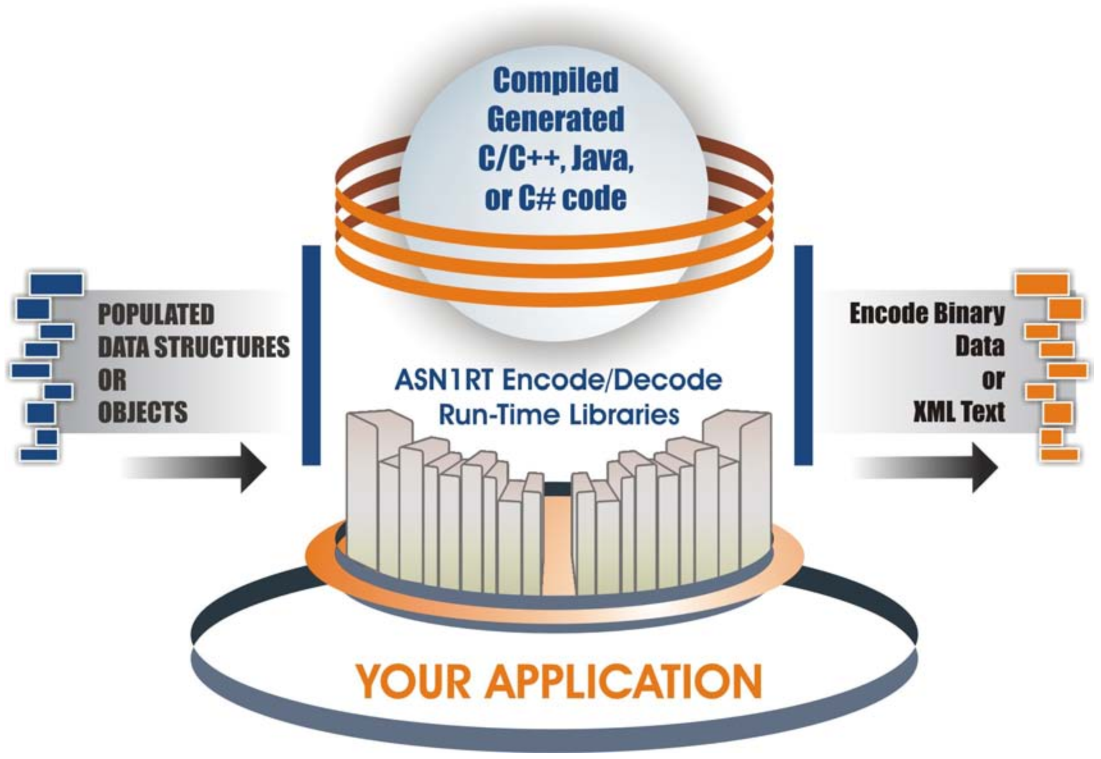




또한 ASN.1 run-time library 를 포함하고 있는데 이것은 컴파일러에 의해 생성된 코드가 ASN.1 메시지의 encoding 과 decoding 을 위해서 완벽한 패키지를 제공합니다. 컴파일러는 여러 종류의 플랫폼에 대한 포팅을 허용할 수 있는 ANSI 표준의 실시간 C 코드를 생성합니다. ASN1CPP 는 ASN.1 의 C++버전 제품입니다. ASN1CPP 역시 동일한 기본 파일 셀과 라이브러리로 동작하며, 또한 encode/decode 함수호출에 대한 정확한 내용을 포함한 wrapper 클래스를 생성합니다. 이는 원래의 C 함수와의 인터페이스 에 관계없이 C++ 환경의 작업을 가능하게 합니다.

다음은 다른 ASN.1 컴파일러 벤더들과 비교되는 특징들입니다.

- 생성된 코드의 사용에 대한 라이센스 비용이 아님
- 다양한 플랫폼에서 사용하기 위한 런타임 라이브러리들을 포함한 소스코드
- ITU X.680 에서 정의한 ASN.1 구문을 파싱하는 능력.
- Object specification (X.681) 과 parameterized type (X.683) 구문에 대한 정보를 지원
- Basic Encoding Rules (BER), Distinguished Encoding Rules (DER) 그리고 Packed Encoding Rules (PER)에 따르는 인코더/디코더들를 생성
- 컴파일러는 또한 더 오래된 x.208 와 x.209 표준들과 하위 상호 교환성을 갖는다.
- ASN.1 to Java or C# 지원합니다.



Objective System ASN1C Version 6.1.3 이 공식발표 되었습니다!

- Visual Studio 2008 지원.
- Java/C# static enumeration 코드 생성 지원.
- Symbian OS 지원.
- X.680 amendment 3(ISO date/time types) 지원.
- ASN1VE ASN.1 (Viewer /Editor) GUI tool 지원(Optional)

- Distinguished Encoding Rules (DER)의 지원
- X.509, PKIX 와 PKCS.같은 ASN.1 의 안전한 스팩을 위한 파싱과 코드생성 RANAP, NBAP 와 RRC 같은 새로운 UMTS 한 3GPP 스팩을 위한 파싱과 코드생성
- Large integers(>32 비트) 지원
- 임베디드 환경을 위한 작고 경제적인 코드생성에 대한 추가적으로 향상된 기능들. Windows, Solaris, Linux 를 위한 Trial version 이 있습니다. 다른 시스템을 위한 버전이 필요하시면 연락 주십시오.

Objective System ASN1C/CPP 특징

- ASN1C 와 ASN1CPP 는 Packed Encoding Rules (PER)을 지원합니다.
- Open Source 런타임 라이브러리 소스코드는 이종의 오퍼레이팅 환경으로 커스터마이징하거나 포팅하는 기능을 제공합니다.
- Standards Based 최신의 ITU ASN.1 표준을 지원합니다. 이전의 표준도 같이 사용가능 합니다.
- Flexible Architecture 생성된 모든 소스와 런타임 코드는 ANSI 표준 C/C++ 코드이므로 다양한 플랫폼으로 포팅할 수 있습니다.
- Easy to Use ASN1C 컴파일러는 ASN.1 의 논리적인 표현과 일치하는 C/C++ 소스코드를 생성합니다
- 64-bit libraries 지원합니다.

1-5-1. Objective System ASN1C Compiler Q&A

ASN1C 와 ASN1CPP 컴파일러에 대한 FAQ 와 그에 대한 해결책:

- **<질문>**ASN1 컴파일러로 생성된 소스코드가 포팅가능 합니까?
- **<회답>**예, 생성된 코드는 ANSI 표준이며, 또한 제품에 포함되어 있는 run-time 라이브러리 소스코드 또한 ANSI 표준입니다. 따라서 완벽한 ANSI 표준 소스코드 패키지이기 때문에 특별히 3rd-party 라이브러리가 필요 없이 다른 플랫폼으로 포팅이 가능합니다.
- **<질문>**Real-time embedded system 의 어플리케이션에서 사용할 수 있는 코드인가요?
- **<회답>**예, 전체 소스코드가 제공되기 때문에 일반적으로 임베디드 환경에 적용시킬 수가 있습니다. 소스코드는 또한 특정의 encoding/decoding 작업을 수행하기 위해서 반드시 필요한 것들을 포함시켜 에디팅 할 수 있습니다. 즉, 가장 공간에 제약적인 시스템에 조차 사용자가 필요한 코드를 생성할 수 있게 합니다.
- **<질문>**어떤 타입의 ASN.1 스팩을 사용할 수 있나요?

- **<회답>**우리의 ASN.1 컴파일러는 표준화 및 관련업계의 특성화된 ASN.1 스팩 모두를 포함하는 다양한 프로젝트에서 사용할 수 있습니다. 표준화된 스팩의 몇 가지 예는 다음과 같습니다.
  UMTS 3GPP Applications
  Computer Supported Telephony Applications (CSTA)
  TAP3
  Mobile Applications Protocol (MAP)
  Aeronautical Telecommunications Network (ATN)
  H.323 VoiP related protocols
  T.38 Real-Time Fax over IP
  ACSE and ROSE

ASN.1C Compiler License QA

다음은 ASN1C 와 ASN1CPP 컴파일러의 라이센스에 관한 Q&A 입니다.

- **<질문>**node-locked 라이센스란 어떤 것인가요?
  <회답>node-locked 라이센스는 제품등록을 할 때 컴파일러를 설치한 컴퓨터 내부의 시스템콜로부터 얻게 되는 노드의 정보들을 가지고 라이센스를 생성하게 됩니다. 따라서, 컴파일러를 설치하는 컴퓨터(node) 에서만 유효하게 동작하는 라이센스 타입을 말합니다.

- **<질문>**node-locked 라이센스는 다른 컴퓨터(node)로 소스를 포팅하는 것에도 제약이 있나요?
  <회답>아니요, 런-타임 라이브러리에는 제약이 없습니다. 구입한 시점에서 컴파일러에 제공되었던 라이센스의 기간 동안에는 다른 OS 환경으로의 포팅이 가능합니다.

- **<질문>**만약에 standard 버전을 구입한다면, 런-타임 라이브러리 소스코드를 우리의 제품에 포함할 수 있나요?
  <회답>재분배 허가(redistribution license)는 무료로 런-타임 라이브러리(소스 코드가 아니라)의 바이러리 버전을 재배포하는 것을 허락합니다. 만약, 귀사의 제품으로 소스 코드를 재배포하고 싶다면, 이것은 라이센스 합의(license agreement)와는 분리된 사항이므로 또 다른 합의가 필요합니다.

- **<질문>**컴파일러에 접근하는 복수의 사용자가 있을 것으로 예상되는 큰 프로젝트가 있습니다. 그렇다면 공유되는 파일 서버에 컴파일러를 셋업할 수 있나요?
  <회답>예, 가능합니다. 그러나 이 경우에는 복수의 사용자 라이센스가 요구됩니다. node-locked 라이센스 파일은 컴파일러가 실행되는 워크스테이션들의 각각의 이름을 포함하는 셋업작업이 필요합니다.

- **<질문>**복수의 사용자(multiple-user) 라이센스를 사면 가격할인이 있나요?
  <회답>예, 할인이 됩니다. 우선 추가적인 라이센스는 basic 라이센스가 됩니다.(만약, 런-타임 소스코드를 제공하는 standard 라이센스로 구입했다면, 나중에 user 의 추가에 따른 라이센스는 basic 라이센스만을 구입하면 됩니다.)추가 라이센스의 가격은 다음과 같습니다 :5 user pack: Price of 2 additional basic licenses.10 user pack: Price of 5 additional basic licenses.Site license (사용자수의 제한이 없슴): full-source version 이 제공됨.

- **<질문>**만약에 컴파일러를 설치할 컴퓨터가 여러 플랫폼이라면 어떻게 구매를 해야 하나요?
  <회답>이 경우는 복수의 사용자(multiple-user)와 비슷한 경우입니다. 만약 standard 버전을 구입한다면, 1 개의 standard 라이센스만 구입하고 나머지는 basic 라이센스로 구입하면 됩니다. 왜냐하면 런-타임 소스코드는 모든 플랫폼에 대하여 동일하기 때문에 오직 하나만

있으면 됩니다. 따라서 처음에 standard 버전을 구입한 후에 각각의 플랫폼에는 basic 라이센스만 하나씩 구입하면 됩니다.

- **<질문>**새로운 버전으로 업그레이드하는 방법은?
  <회답>마이너 패치버전의 제품(Minor patch releases) 기술지원 서비스(support period) 기간이 지나도 항상 무료로 보내 드립니다. 패치 릴리즈는 버전 번호중에 소수점 두번째 자리의 변화를 의미합니다(예를들어 . v5.01, v5.02,등). 한편, 주요한 기능의 업그레이드(예를들어 5.1x or 6.1x 등)는 귀사가 기술지원 서비스 기간이 지나지 않았다면 가능합니다. 그러나 이것이 무상 업그레이드를 의미하는 것은 아닙니다. 일반적으로 업그레이드 금액은 새로운 버전은 기능이 더 나아지기 때문에 옛 버전보다 더 비싼 가격이 되고, 따라서 옛 버전과 새 버전의 차액만을 지불하면 됩니다. 만약에 기술지원 서비스 계약기간이 지난 후에 업그레이드를 하게 되면, 50%의 가격할인을 적용 받게 됩니다.

- **<질문>**기술지원 서비스 계약기간을 연장할 수는 있나요?
  <회답>예, 원래 제품가격의 20% 비용을 지불하면 1 년간 계약기간을 연장할 수 있습니다.

- **<질문>**다른 회사의 제품보다 더 가격이 낮은 이유는 뭔가요?
  <회답>우리는 정직한 평가모델을 바탕으로 제품의 가격을 산정합니다. 우리의 목표는 공정한 가격과 좋은 기능의 컴파일러를 제공하는 것입니다. 가격이 다른 회사의 제품보다 싸게 판다고 해서 품질이나 서비스의 질이 떨어지는 것은 절대 아닙니다. 그것을 확인하기 위해서 평가판을 자유롭게 다운 받아서 사용해 보시기 바랍니다. 우리는 고객들이 제품들을 비교해서 자신에게 어떤 제품이 더 가치가 있는지를 판단해 주기를 바랍니다.

2-1. ASN1C ASN.1 Compiler Software 설치

Version 6.1x of the ASN1C compiler software is packaged in two separate distribution units:

- A System Development Kit (SDK) unit that contains the compiler and development libraries, and
- One or more run-time deployment units containing optimized binary libraries for deployment of a finished application.

2-1-1 Microsoft Windows Distribution

The Microsoft Windows version of ASN1C is distributed either on a CD and/or electronically over the Internet. The distribution files are self-extracting, executable setup files. The format of the filename of the SDK unit is as follows:

```
ac<L>v62xw32sdk.exe
```

where *<L>* would be replaced with a single-letter language code. Possible values are *p* for C/C++, *j* for Java, or *s* for C#.
Run-time deployment packages have the following format:

```
rt<L>v62xw32<TYP>.exe
```

where *<L>* would be replaced with a single-letter language code and *<TYP>* would be replaced with a code for the package type. Possible values for the language code are *p* for C/C++, *j* for Java, or *s* for C#.
The type code would contain information on whether the library is licensed per-host (limited) or unlimited, whether source code is included or not, and what encoding rules are supported.
For example: the following would be an ASN1C v6.10 run-time deployment kit for Java:

```
rtjv620w32ubb.exe
```

In this case, the *ubb* on the end stands for unlimited, binary, BER/DER.

ASN1C System Development Kit (SDK) Installation

The procedure to install the ASN1C compiler and run-time libraries is as follows:

1. Double-click the SDK installation program kit executable filename. This is the filename in the format described above.

2. Follow the setup program instructions.

3. You should have received a node-locked license file (*osyslic.txt*) to enable the compiler to run on a given node. Copy this file into one of the following locations:

a. One of the directories specified in your PATH environment variable, or
b. In a different directory and create a new environment variable name OSLICDIR that points to this directory, or
c. Into the same directory as the ASN1C compiler executable file (asn1c.exe)

If installation was successful, you should have both a graphical user interface (GUI) compilation wizard
available as well as a command line version of the compiler. The GUI wizard can be tested by starting the
application and entering the data requested in each of the dialog prompts. The command-line version of the
compiler can be tested as follows:

1. Open an MS-DOS or other command shell window.

2. Change directory (cd) to the compiler root directory. The default directory in the setup script is
   c:₩acv<version> where <version> is the version number of the compiler. For example, c:₩acv620 is the default root directory for version 6.20 of the compiler.

3. Enter .₩bin₩asn1c from the command line prompt. You should see a usage display of compiler command line options (these options are discussed later). This indicates the compiler is properly installed and working.

If you get a message indicating the license file could not be found, please review the procedure in step 3 above to make sure it is installed in the proper location.

You should include the <target>₩bin (ex. c:₩acv610₩bin) directory in your PATH environment variable in order to run the compiler from anywhere.

ASN1C Run-time Deployment Kit Installation

The deployment run-time packages can be installed at any time after the SDK is installed. They are not necessary for basic program development. They should be used when an application is ready to be deployed. To install, do the following:

1. Double-click the installation program kit executable filename. This is the filename in the format described above.

2. Follow the setup program instructions. Note that the root directory for the installation should be the same as was specified for installation of the SDK package described earlier.

The result will be additional library subdirectories of the form *lib_opt* added to the directory hierarchy.
These contain the optimized libraries. To link with these libraries, either the makefile(s) or project files used to link the application must be changed, or the subdirectories must be renamed. For example, the existing *lib* subdirectory could be renamed *lib_nonopt* and *libopt* could then be renamed to *lib*.

Documentation Installation

Up-to-date documentation is always available online at:
http://www.obj-sys.com/docs/documents.shtml

If a CD was purchased with the software, all applicable PDF document files for a particular configuration of the software will be available on the CD.

Compiling and Linking ASN1C Generated Code

When building code generated by the ASN1C compiler, you will need to have one or more run-time source directories in your include path to compile the generated code with a C or C++ compiler. The run-time source directories are *rtsrc* (common), *rtbersrc* (BER/DER), *rtpersrc* (PER), and *rtxersrc* (XER). To link, you will need the *lib* or *lib_opt* subdirectory in your library path. This is where all of the library files are located.

Testing the C or C++ Run-time Components

The default C or C++ run-time libraries for Windows were built with Microsoft Visual C++ V6.0. Other libraries are available that have been built with the Borland C++ compiler (v5.5) and with the Microsoft Visual C++ v7.1, v8.0, and v9.0 (.NET) compilers. If you have the version of the product that includes run-time source code, you can rebuild the run-time libraries using any ANSI-standard C or C++ compiler.

You can verify operation of any of the different run-time libraries by executing the sample programs. These can be found in the *sample_ber*, *sample_der*, and (optionally) the *sample_per* or *sample_xer* subdirectories.

For example, we will assume that you installed ASN1C for C/C++. To test the BER C++ encode/decode capabilities, do the following:

1. Change directory (cd) to the **.₩cpp₩sample_ber₩employee** subdirectory. Execute the **nmake** command to build the writer and reader sample programs. **nmake** is utility program that comes with Visual C++. It may also be necessary to execute a Microsoft batch file named **VCVARS32.BAT** to set the path information so that the **nmake** utility can be found. (Important note: this assumes you are using Microsoft Visual C++ on your PC. Some PC specific include and library directories in the makefile may need to be changed to get the samples to work on your system. See the README.txt file for further details).

2. Execute the **writer.exe** program to encode the sample record. The results of the encoding will be dumped to the screen and saved in a file called *message.dat*.

3. Execute the **reader.exe** program to read and decode the contents of the *message.dat* file. This program will read the encoded record into memory, decode it, and then print the contents of the generated structure variable to standard output.

Testing PER is similar:

1. Change directory (cd) to the **.₩sample_per₩employee** subdirectory. Execute the **nmake** command to build the writer and reader sample programs.

2. Execute the **writer.exe** program to encode the sample record. The **.a** switch can be used to encode a record using aligned PER. The **.u** switch encodes a record using unaligned PER. The results of the encoding will be dumped to the screen and saved in a file called *message.dat*.

3. Execute the **reader.exe** program to read and decode the contents of the *message.dat* file. The **.a** or **.u** switch must be the same as that specified when the writer program was executed. This program will read the encoded record into memory, decode it, and then print the contents of the generated structure variable to standard output. This test can be repeated for XER as well by going to the **sample_xer₩employee** subdirectory and repeating the above sequence of steps.

Per-host License Deployment Issues

If you purchased run-time libraries that allow for unlimited redistribution, then all run-time license checking would have been removed from these libraries and all that must be done to deploy is to make certain that the code is linked with these libraries.
If per-host run-time licensing was selected, then there are two choices as to how applications are deployed:

1. The run-time license information can be directly compiled into the application and then the application deployed without any external files, or

2. An external binary license file can be deployed with the application to allow it to run on the licensed hosts.

The first choice is done by default every time ASN.1 source files are generated and the resulting code compiled and linked into an application. Information from the *osyslic.txt* file is transferred to a C header file called *rtkey.h* and this information is included in the generated code. This allows the application to run on all licensed hosts.

The second option is applicable mainly in situations when a finished application is to be run on different hosts than were originally licensed. This can happen if host names are changed, or additional hosts are added at a later date. It might not be practical in these situations to rebuild the application in this case. The alternative is therefore available to create a binary license file and deploy it with the application to allow it
to run on the newly licensed hosts. The procedure to do this is as follows:

1. Use ASN1C to generate an *rtkey.dat* file by issuing the following command:
   asn1c –genlic

The *rtkey.dat* file will be created in the directory where you issued the command.

2. Copy the *rtkey.dat* file onto the computers on which you want to run your application.

3. Set the *ACLICFILE* environment on these computers to point at the full path to the *rtkey.dat* file.

In the case of Java, the procedure is different. In this case, the deployed *asn1rt.jar* file must contain up-todate license information. This is done by executing the *setkey.bat* script for Windows or *setkey.sh* script for Linux/UNIX in the Java subdirectory after the license file for ASN1C is installed. This must be done whenever the *osyslic.txt* license file is updated.

2-1-2. UNIX Distribution

Installation of the Linux / UNIX version of ASN1C is similar to the Windows version except that the distribution files are packaged as gzipped tar files. The format is the same as Windows except that the extension is *.tar.gz* instead of *.exe.* To install, do the following:

1. Copy the distribution file <distfile> to the top-level directory where the compiler is to be installed.

2. Unzip using the GNU unzip tool:
   ```
   gunzip <distfile>
   ```

3. Untar the file using the following command:
   ```
   tar xvf <distfile>
   ```
   This will create a directory tree structure with ‘ asn1c-v<version>’ as the root.

4. You should have received a node-locked license file (*osyslic.txt*) to enable the compiler on a given node. Copy this file into one of the following locations:

   a. One of the directories specified in your PATH environment variable, or
   b. In a different directory and create a new environment variable name OSLICDIR that points to this directory, or
   c. Into the same directory as the ASN1C compiler executable file (asn1c.exe)

To test if the compiler installation was successful, do the following:

1. Change directory (cd) to the compiler root directory. The default directory in the setup script is
   ./asn1c-v<version> where **<version>** is the version number of the compiler. For example, **./asn1c-v612** is the default root directory for version 6.10 of the compiler.

2. Enter **./bin/asn1c** from the command line prompt. You should see a usage display of compiler command line options (these options are discussed later). This indicates the compiler is properly installed and working.

After installing the compiler, you can modify your operating environment to access the compiler executable files from anywhere. To do this, you will need to add the path to the compiler to your

PATH environment variable or add a link to it from a standard binary directory (such as /opt/bin). This is optional and not required for any of the run-time sample programs to work. They are all set up to use relative directory paths to access the compiler and libraries.

Deployment run-time packages should be installed by repeating steps 1 through 3 above. The packages should be unpacked in the same base root directory as the original SDK files. This will cause *lib_opt* subdirectories to be added to the various C and C++ directories in the installation.

Testing the C or C++ Run-time Components

The basic C or C++ run-time libraries for UNIX are typically built with the GNU gcc/g++ compiler and/or the standard native compiler provided by the manufacturer of a particular type of UNIX (for example, aCC for HP-UX). Two symbolic links are used within the c or cpp subdirectory to select the version of the runtime libraries to be used. They are as follows:

- lib
- platform.mk

By default, these are set to point at the GNU gcc/g++ version of the run-time libraries for a particular platform. This is easily changed by deleting the links and setting them to point at another run-time library. For example, on Solaris, to use the native compiler libraries one would set lib -> libCC and platform.mk -> platform.CC:

```
ln .s libCC lib
ln .s platform.CC platform.mk
```

You can verify operation of any of the different run-time kits by executing the sample programs. These can be found in the different sample directories (*sample_ber*, *sample_der*, *sample_per* and/or *sample_xer* depending on what run-time kits were installed).

To test the encode/decode capabilities for any of the encoding rules, execute the sample programs as described in the section on Windows installation.

Backwards Compatibility

ASN1C underwent significant changes between version 5.8x and 6.0x, including some that affect code generation. These changes are also present in version 6.1. Additionally, 6.1 includes more modifications that will affect Java and C# handling of enumeration types, especially.
These changes represent an API change for all languages. For C and C++, two files have been packaged with the kit to help in porting code generated with prior versions: rtport.pl and asn1compat.h. The former will process a specified file and rename the types contained in it. The latter contains macro definitions that convert old names to the new naming scheme.
Changes for Java and C# will be mostly transparent to the user, except that enumerated values are now retrieved through the use of accessor methods. These changes are discussed in detail in the Java and C#
User's Guides.

2-2. COMMAND-LINE ASN1C 실행

(Getting Started with the ASN1C C / C++ command line.)

Getting Started with the ASN1C C / C++ Command Line

This tutorial will help users use the ASN1C complier using the terminal shell prompt. Please note that tutorial assumes you are operating in a Linux system.

Change directory to one of the employee sample directories.

```
cd c/sample_ber/employee
```

Change the PATH variable by typing:

```
PATH=$HOME/asn1c-v60x/bin:$PATH
export PATH
```

Note: This assumes ASN1C was installed into the login root directory. Also, x is to be replaced by a minor version number.

Run ASN1C to compile the files.

To compile the included employee example, type the following on the command line:

For C:

```
asn1c employee.asn -c -ber
```

For C++:

```
asn1c employee.asn -c++ -ber
```

Run the C / C++ compiler.

Note: This tutorial shows how to use the GNU gcc/g++ compiler, but the procedure for other compilers would be similar.

At a minimum, the -c and -I options are necessary to compile the employee.c (or employee.cpp) file. -I is used to specify include directories. For BER, the rtsrc and rtbersrc directories would need to be included, therefore, the command line argument would be:

For C:

```
gcc -c -I $(HOME)/asn1c-v60x/rtsrc -I $(HOME)/asn1c-v60x/rtbersrc employee.c
```

For C++:

```
g++ -c -I $(HOME)/asn1c-v60x/rtsrc -I $(HOME)/asn1c-v60x/rtbersrc employee.cpp
```

The same command would be required to compile the reader.c and writer.c programs.

Link the compiled code with the runtime libraries.

For C:

```
gcc -o writer writer.o employee.o -L$(HOME)/asn1c-v60x/c/lib -lasn1ber -lasn1rt
gcc -o reader reader.o employee.o -L$(HOME)/asn1c-v60x/c/lib -lasn1ber -lasn1rt
```

For C++:

```
g++ -o writer writer.o -L$(HOME)/asn1c-v60x/cpp/lib -lasn1ber -lasn1rt
g++ -o reader reader.o -L$(HOME)/asn1c-v60x/cpp/lib -lasn1ber -lasn1rt
```

Note: A makefile and a Visual Studio project file (Microsoft Windows only) contain all of these commands. These files come with the ASN1C compiler.

2-3 ASN1C Compiler Option

```
ASN1C Compiler, Version 6.2.x
Copyright (c) 1997-2009 Objective Systems, Inc. All Rights Reserved.

Usage: asn1c <filename> <options>

    <filename>            ASN.1 or XSD source filename(s).  Multiple filenames
                          may be specified.  * and ? wildcards are allowed.

  language options:
    -c                  generate C code
    -c++                generate C++ code
    -c#                 generate C# code
    -java               generate Java code
    -xsd [<filename>]   generate XML schema definitions

  encoding rule options:
    -ber                generate BER encode/decode functions
    -cer                generate CER encode/decode functions
    -der                generate DER encode/decode functions
    -per                generate PER encode/decode functions
    -xer                generate XER encode/decode functions
    -xml                generate XML encode/decode functions

  basic options:
    -asnstd <std>       set standard to be used for parsing ASN.1
                        source file. Possible values - x208, x680, mixed
                        (default is x680)
    -compact            generate compact code
    -compat <version>   generate code compatible with previous
                        compiler version. <version> format is
                        x.x (for example, 5.3)
    -config <file>      specify configuration file
    -depends             compile main file and dependent IMPORT items
    -I <directory>       set import file directory
    -lax                 do not generate constraint checks in code
    -laxsyntax           do not do a thorough ASN.1 syntax check
    -list                generate listing
    -noContaining        do not generate inline type for CONTAINING <type>
    -nodecode            do not generate decode functions
    -noencode            do not generate encode functions
    -noIndefLen          do not generate indefinite length tests
    -noObjectTypes       do not gen types for items embedded in info objects
    -noOpenExt           do not generate open extension elements
    -notypes             do not generate type definitions
    -noxmlns             do not generate XML namespaces for ASN.1 modules
    -o <directory>       set output file directory
    -pdu <type>          designate <type> to be a Protocol Data Unit (PDU)
                        (<type> may be * to select all type definitions)
    -usepdu <type>       specify a Protocol Data Unit (PDU) type for which
                        sample reader/writer programs and test code has to
                        be generated
    -print [<filename>] generate print functions
    -prtfmt details | bracetext  format of output generated by print
    -shortnames          reduce the length of compiler generated names
    -trace               add trace diag msgs to generated code
    -[no]UniqueNames     resolve name clashes by generating unique names
                          default=on, use -noUniqueNames to disable
    -warnings            output compiler warning messages
    -nodatestamp         do not put date/time stamp in generated files

  C/C++ options:
    -hfile [<filename>]    C or C++ header (.h) filename
                          (default is <ASN.1 Module Name>.h)
    -cfile [<filename>]    C or C++ source (.c or .cpp) filename
                          (default is <ASN.1 Module Name>.c)
    -genBitMacros        generate named bit set, clear, test macros
    -genFree             generate memory free functions for all types
    -hdrGuardPfx         add prefix to header guard #defines in .h files
    -maxlines <num>      set limit of number of lines per source file
                        (default value is 50000)
    -noInit              do not generate initialization functions
    -oh <directory>      set output directory for header files
    -static              generate static elements (not pointers)
    -cppNs <namespace>   add a C++ namespace to generated code (C++ only)
  C/C++ makefile/project options:
    -genMake [<filename>] generate makefile to build generated code
    -vcproj [<version>]   generate VC++ 6.0 project files for use with <version> (Windows
only)
    -dll                  generate makefile/project to use DLL's
    -mt                   generate makefile/project to use multithreaded libs
    -w32                  generate code for Windows O/S (default=GNU)

  Java options:
    -compare          generate comparison functions
    -dirs             output Java code to module name dirs
    -genbuild         generate build script
    -genant           generate ant build.xml script
    -genjsources      generate <modulename>.mk for list of java files
    -getset           generate get/set methods and protected member vars
    -pkgname <text>   Java package name
    -pkgpfx <text>    Java package prefix
    -java4            generate code for Java 1.4

  C# options:
    -nspfx <text>     C# namespace prefix
    -namespace <text> C# namespace name
    -dirs             output C# code to module name dirs
    -gencssources     generate <modulename>.mk for list of C# files
    -genMake          generate makefile to build generated code

  pro options:
    -3gpp             generate special code for 3GPP specifications
    -events           generate code to invoke SAX-like event handlers
    -stream           generate stream-based encode/decode functions
    -tables           generate table constraint functions
    -strict           do strict checking of table constraint conformance
    -param <name>=<value> create types from param types using given value
    -prtToStr [<filename>]
                      generate print-to-string functions (C/C++)
    -prtToStrm [<filename>]
                      generate print-to-stream functions (C/C++)
    -genTest  [<filename>]
                      generate sample test functions
    -domTest  [<filename>]
                      generate test functions that use XML DOM
    -reader            generate sample reader program
    -writer            generate sample writer program
    -compare [<filename>]
                      generate comparison functions (C/C++)
    -copy [<filename>] generate copy functions (C/C++)
    -maxcfiles         generate separate file for each function (C/C++)

  XSD options:
    -appinfo [<items>] generate appInfo for ASN.1 items
                         <items> can be tags, enum, and/or ext
                         ex: -appinfo tags,enum,ext
                         default = all if <items> not given
    -attrs [<items>]   generate non-native attributes for <items>
                         <items> is same as for -appinfo
    -targetns [<namespace>] Specify target namespace
                        <namespace> is namespace URI, if not given
                        no target namespace declaration is added
    -useAsn1Xsd         reference types in asn1.xsd schema

  Symbian options:
    -symbian [<items>] generate code for Symbian OS
                         <items> can be dll
                         e.g. -symbian dll
                         default = symbian application style code
```

2-2-1. nmake clean

```
C:\acv580\cpp\sample_ber\employee>nmake clean

Microsoft (R) Program Maintenance Utility   Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

        del Employee*.cpp
        del Employee.h
        del *.obj
        del *.exe
        del *.exp
        del *.pdb
        del *.map
        del *.lib
        del *~
        del writer.exe
        del reader.exe
        del rtkey.h

C:\acv580\cpp\sample_ber\employee>dir

2005-11-09  오후 05:03               545 employee.asn
2005-11-09  오후 05:03               880 employee.dsw
2006-04-18  오후 04:19            50,176 employee.ncb
2006-04-18  오후 04:19            62,976 employee.opt
2005-11-09  오후 05:03             1,310 makefile
2006-04-18  오후 04:17               136 message.dat
2005-11-18  오후 01:40               161 messagei.dat
2005-11-09  오후 05:03             2,815 reader.cpp
2005-11-09  오후 05:03             5,728 reader.dsp
2006-04-18  오후 04:14             9,539 reader.plg
2006-04-18  오후 04:14    <DIR>          ReaderDebug
2006-04-18  오후 04:13    <DIR>          ReaderRelease
2005-11-09  오후 05:03             3,846 writer.cpp
2005-11-09  오후 05:03             5,623 writer.dsp
2006-04-18  오후 04:14               246 writer.plg
2006-04-18  오후 04:14    <DIR>          WriterDebug
2006-04-18  오후 04:14    <DIR>          WriterRelease
```

2-2-1. nmake

```
C:\acv580\cpp\sample_ber\employee>nmake

Microsoft (R) Program Maintenance Utility      Version 6.00.8168.0
Copyright (C) Microsoft Corp 1988-1998. All rights reserved.

        ..\\..\\..\\bin\\asn1c employee.asn -c++ -ber -trace -print -geninit

ASN1C Compiler, Version 5.80
Copyright (c) 1997-2005 Objective Systems, Inc. All Rights Reserved.

License expire date is Sat May 20 21:26:27 2006

Parsing ASN.1 definitions..

Writing C++ class definitions to file Employee.h..
Writing C++ common definitions/functions to file Employee.cpp..
Writing C++ encode functions to file EmployeeEnc.cpp..
Writing C++ decode functions to file EmployeeDec.cpp..
Writing C++ common definitions/functions to file Employee.cpp..
Writing C++ print functions to file EmployeePrint.cpp..

        cl -D_TRACE -Ob2 -Gy -Ox -G6 -D_OPTIMIZED -DWIN32 -D_WIN32 -
DMSDOS -GF -ML    -nologo -GX -W3    -c -I. -I../../src
-I../../../rtsrc -I../../../rtbersrc -I../../../rtpersrc -I../../../rtxersrc    -I../../..    writer.cpp
writer.cpp

        cl -D_TRACE -Ob2 -Gy -Ox -G6 -D_OPTIMIZED -DWIN32 -D_WIN32 -
DMSDOS -GF -ML    -nologo -GX -W3    -c -I. -I../../src
-I../../../rtsrc -I../../../rtbersrc -I../../../rtpersrc -I../../../rtxersrc    -I../../..    Employee.cpp
Employee.cpp

        cl -D_TRACE -Ob2 -Gy -Ox -G6 -D_OPTIMIZED -DWIN32 -D_WIN32 -
DMSDOS -GF -ML    -nologo -GX -W3    -c -I. -I../../src
-I../../../rtsrc -I../../../rtbersrc -I../../../rtpersrc -I../../../rtxersrc    -I../../..    EmployeeEnc.cpp
EmployeeEnc.cpp

        cl -D_TRACE -Ob2 -Gy -Ox -G6 -D_OPTIMIZED -DWIN32 -D_WIN32 -
DMSDOS -GF -ML    -nologo -GX -W3    -c -I. -I../../src
-I../../../rtsrc -I../../../rtbersrc -I../../../rtpersrc -I../../../rtxersrc    -I../../..    EmployeeDec.cpp
EmployeeDec.cpp

        cl writer.obj Employee.obj EmployeeEnc.obj EmployeeDec.obj    /nologo /link
/OUT:writer.exe /OPT:REF -LIBPATH:../.
./lib    asn1ber_a.lib asn1rt_a.lib user32.lib ws2_32.lib advapi32.lib
```

```
        cl -D_TRACE -Ob2 -Gy -Ox -G6 -D_OPTIMIZED -DWIN32 -D_WIN32 -
DMSDOS -GF -ML    -nologo -GX -W3    -c -I. -I../../src
-I../../../rtsrc -I../../../rtbersrc -I../../../rtpersrc -I../../../rtxersrc    -I../../..    reader.cpp
reader.cpp

        cl -D_TRACE -Ob2 -Gy -Ox -G6 -D_OPTIMIZED -DWIN32 -D_WIN32 -
DMSDOS -GF -ML    -nologo -GX -W3    -c -I. -I../../src
-I../../../rtsrc -I../../../rtbersrc -I../../../rtpersrc -I../../../rtxersrc    -I../../..    EmployeePrint.cpp
EmployeePrint.cpp

        cl reader.obj Employee.obj EmployeeEnc.obj EmployeeDec.obj
EmployeePrint.obj /nologo /link /OUT:reader.exe /OPT
:REF -LIBPATH:../../lib    asn1ber_a.lib asn1rt_a.lib user32.lib ws2_32.lib advapi32.lib

C:\acv580\cpp\sample_ber\employee>dir
2006-04-19  오후 10:02    <DIR>          .
2006-04-19  오후 10:02    <DIR>          ..
2005-11-09  오후 05:03               545 employee.asn
2006-04-19  오후 10:02            10,170 Employee.cpp
2005-11-09  오후 05:03               880 employee.dsw
2006-04-19  오후 10:02             6,985 Employee.h
2006-04-18  오후 04:19            50,176 employee.ncb
2006-04-19  오후 10:02            15,605 Employee.obj
2006-04-18  오후 04:19            62,976 employee.opt
2006-04-19  오후 10:02            12,831 EmployeeDec.cpp
2006-04-19  오후 10:02             9,511 EmployeeDec.obj
2006-04-19  오후 10:02             7,011 EmployeeEnc.cpp
2006-04-19  오후 10:02             6,338 EmployeeEnc.obj
2006-04-19  오후 10:02             4,839 EmployeePrint.cpp
2006-04-19  오후 10:02             6,118 EmployeePrint.obj
2005-11-09  오후 05:03             1,310 makefile
2006-04-18  오후 04:17               136 message.dat
2005-11-18  오후 01:40               161 messagei.dat
2005-11-09  오후 05:03             2,815 reader.cpp
2005-11-09  오후 05:03             5,728 reader.dsp
2006-04-19  오후 10:02           139,264 reader.exe
2006-04-19  오후 10:02            14,973 reader.obj
2006-04-18  오후 04:14             9,539 reader.plg
2006-04-18  오후 04:14    <DIR>          ReaderDebug
2006-04-18  오후 04:13    <DIR>          ReaderRelease
2006-04-19  오후 10:02               853 rtkey.h
2005-11-09  오후 05:03             3,846 writer.cpp
2005-11-09  오후 05:03             5,623 writer.dsp
```

```
2006-04-19  오후 10:02           131,072 writer.exe
2006-04-19  오후 10:02            17,309 writer.obj
2006-04-18  오후 04:14               246 writer.plg
2006-04-18  오후 04:14    <DIR>          WriterDebug
2006-04-18  오후 04:14    <DIR>          WriterRelease
2006-04-19  오후 09:38             1,789 명령 프롬프트.lnk
              28개 파일             528,649 바이트
               6개 디렉터리  10,303,574,016 바이트 남음
```

Visual Studio Projects

(Using the existing Visual Studio project files.)

ASN1C comes complete with several samples, each of which includes a Visual Studio project file compatible with Visual Studio 6.0. (These project files may be imported into newer versions of Visual Studio.) ASN1C versions 6.0 and above also include options to generate project files.

This tutorial describes creating a project file for the **Employee** sample program found in ASN1C_INSTALL_DIR\cpp\sample_ber\employee. On typical Windows installations, ASN1C_INSTALL_DIR is c:\acv[version], where [version] corresponds to the three digit version number, such as 585.

Please note: projects imported into Visual Studio .NET 2005 may compile with several warnings because of deprecated functions. Newer versions of ASN1C generate code that will not cause these warnings.

Following are instructions for using the enclosed project files.

1. Before working with the project, the source code for it must be generated, either using the command-line tool or the graphical user interface. Click here for instructions for using the GUI.
2. Open Microsoft Visual C++.
3. From the File menu, choose Open Workspace.
4. Open the file employee.dsw.
5. reader and writer classes classes should be visible in the project workspace. If you are unable to see the Project workspace, click on the View menu, and select the Workspace option.
6. From the workspace window, right click on the writer class and select Set As Active Project.
7. Select Build writer.exe from the Build menu. This action will compile the writer executable.
8. From the Build menu, select the Execute writer.exe option. An MS-DOS prompt window will open, and you will see the contents of the execution. Close this window by pressing any key.
9. Repeat this process for the reader project.

Creating a Project

(Creating a new Visual Studio project in a sample directory.)

This tutorial will help guide you to complete a Microsoft Visual C++ 6.0 project. The results will enable to build and run the files generated by the ASN1C compiler.

Note: Before you begin to create a project, it is necessary to have first generated the .c / .cpp and .h using the ASN1C GUI or command line.

Open Microsoft Visual C++ .

From the File menu, select select the New option.

A new window will appear on your screen enabling you to create a new workspace, project, or document.

Select the Project tab from the top of the new window.

Select Win 32 Console Application from the list and proceed with project creation wizard.

Enter VSproj for the Project name, and Click the OK button. From the next window, ensure you Create a Blank Project is selected and click Finish.

Once you click the Finish button in the wizard, you will return to the original Visual C++ window.

Once you return to the main window, add the appropriate files to the project.

To add files to a project, select the Project option from the menu. From the pop-up menu, select Add to Project followed by selecting Files from the next pop-up menu. At this point a new window will appear. From this window, select the .c, .cpp, and .h files generated by the ASN1C compiler. For example, select employee.c and employee.h which have been compiled by the ASN1C compiler. When you have selected all of these files, click on the OK button.

NOTE: It is necessary to add both the .c or .cpp file AND the .h file.

From the Project menu, select the Settings option.

This will open a new window allowing you to alter the setting of the current project.

Select the Link tab from the top of the Project Settings window.

Add asn1rt.lib and and one of the following libraries to the Object/library modules text area. Once complete, select OK.

- asn1BER.lib (if BER or DER encoding rules were selected)
- asn1PER.lib (if PER encoding rules were selected)
- asn1XER.lib (if XER encoding rules were selected)

From the Tools menu, select Options.

This will open a new window allowing you to modify the options of your project.

Inside the Directories tab, ensure you are viewing the directories of the Include files

Add C:\ACV58x\RTSRC and one of the following directories to the list of directories:

- C:\ACV58x\RTBERSRC (if BER or DER encoding rules were selected)
- C:\ACV58x\RTPERSRC (if PER encoding rules were selected)
- C:\ACV58x\RTXERSRC (if XER encoding rules were selected)

and the directory of the the library files.

Add C:\ACV58x\c\lib if your sample application is a in C
or C:\ACV58x\cpp\lib if your sample application is a in C++.

NOTE: x is the version number of your ASN1C compiler.

You can now Build and Compile the code.

- Select Build form the Build menu option.
- Select Compile from the Build menu option.

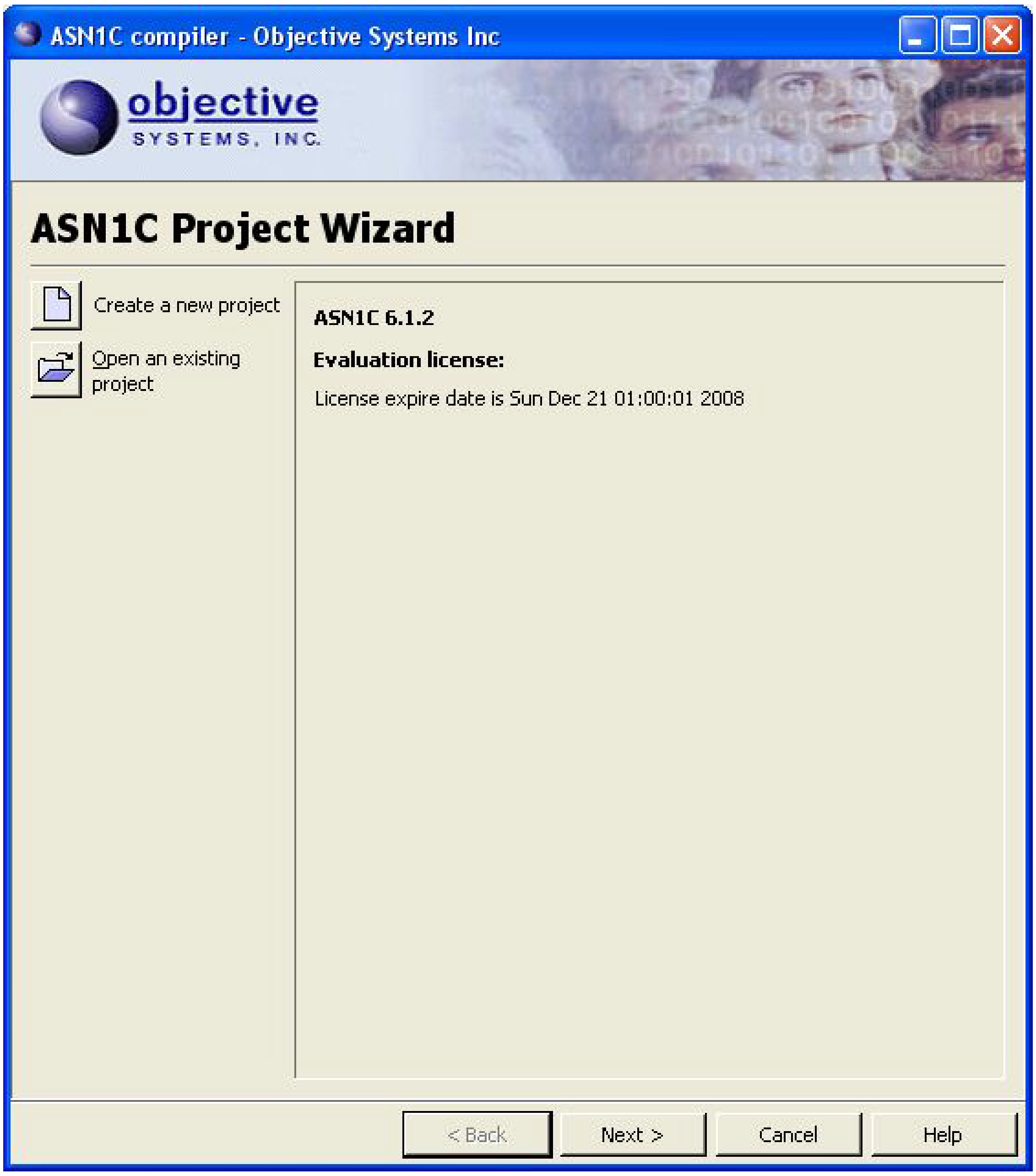
2-3. GUI WIZARD ASN1C 실행

(Getting Started with the ASN1C Graphical User Interface (GUI) Wizard.)

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 1

Double-click on the ASN1C complier icon.

A window will open welcoming you to the ASN1C compiler wizard.



ASN1C 6.1.2

Evaluation license:

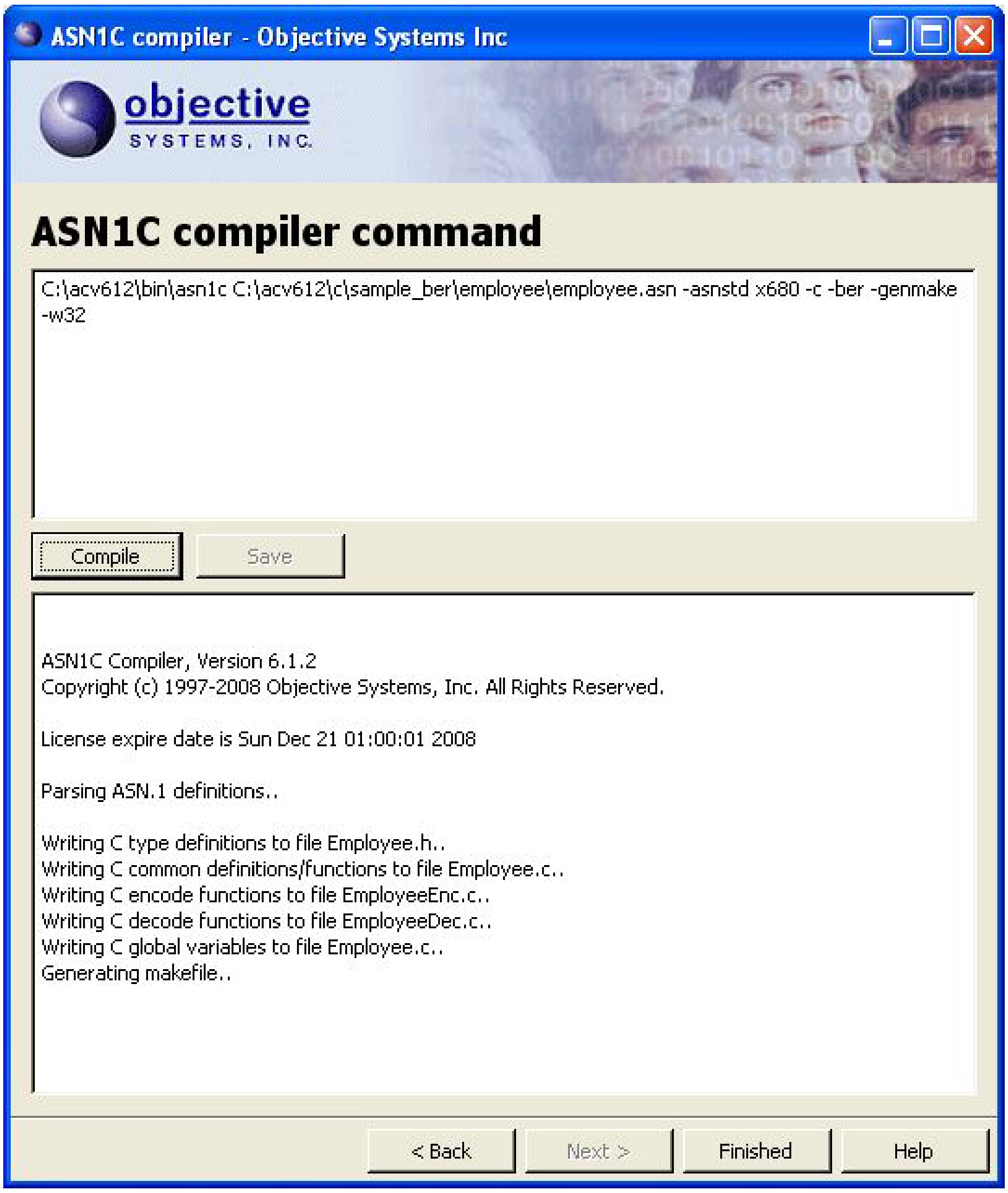
License expire date is Sun Dec 21 01:00:01 2008

Click on the next button located at the bottom of the window.

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 2

In the ASN.1 Files grouping, click on the Add button.

A new window will appear allowing you to select the ASN.1 file you want to translate.



ASN1C compiler command

```
C:\acv612\bin\asn1c C:\acv612\c\sample_ber\employee\employee.asn -asnstd x680 -c -ber -genmake
-w32
```

```
ASN1C Compiler, Version 6.1.2
Copyright (c) 1997-2008 Objective Systems, Inc. All Rights Reserved.

License expire date is Sun Dec 21 01:00:01 2008

Parsing ASN.1 definitions..

Writing C type definitions to file Employee.h..
Writing C common definitions/functions to file Employee.c..
Writing C encode functions to file EmployeeEnc.c..
Writing C decode functions to file EmployeeDec.c..
Writing C global variables to file Employee.c..
Generating makefile..
```

To open the Employee example, click on employee.asn file, and press the Open button.

If you installed the ASN1C compiler to the default location, the employee.asn file is located in C:\acv61x\c\sample_ber\employee directory. Once you select the Open button, the file open window will close and you will return to the ASN1C wizard. C:\acv61x\c\sample_ber\employee\employee.asn is now visible in the ASN.1 Files text window.

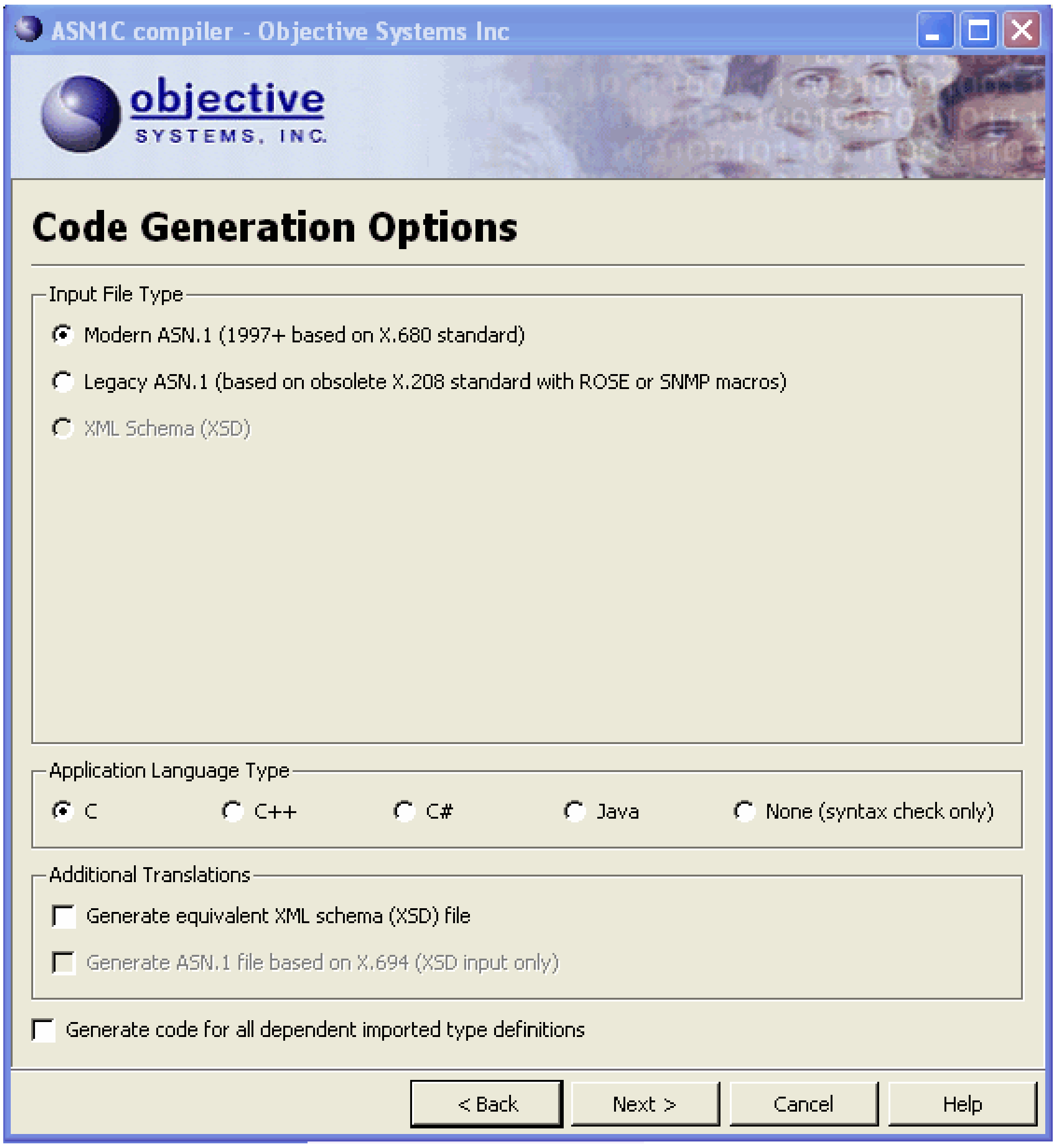
Select the Next button located at the bottom of the window.

It is not necessary to add any directories or files to the Include Directories or Configuration Files sections before clicking on the Next button. If you want to include this information for the ASN1C compiler, follow the same process listed above for the ASN.1 Files.

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 3

By default, the 1997/2002 ASN.1 Standard category is chosen. It is not necessary to change this to continue with the ASN1C wizard.

Choose an application language from the center of the next page.



Code Generation Options

Input File Type

- [✓ Modern ASN.1 (1997+ based on X.680 standard)]
- Legacy ASN.1 (based on obsolete X.208 standard with ROSE or SNMP macros)
- XML Schema (XSD)

Application Language Type

[✓ C] C++ C# Java None (syntax check only)

Additional Translations

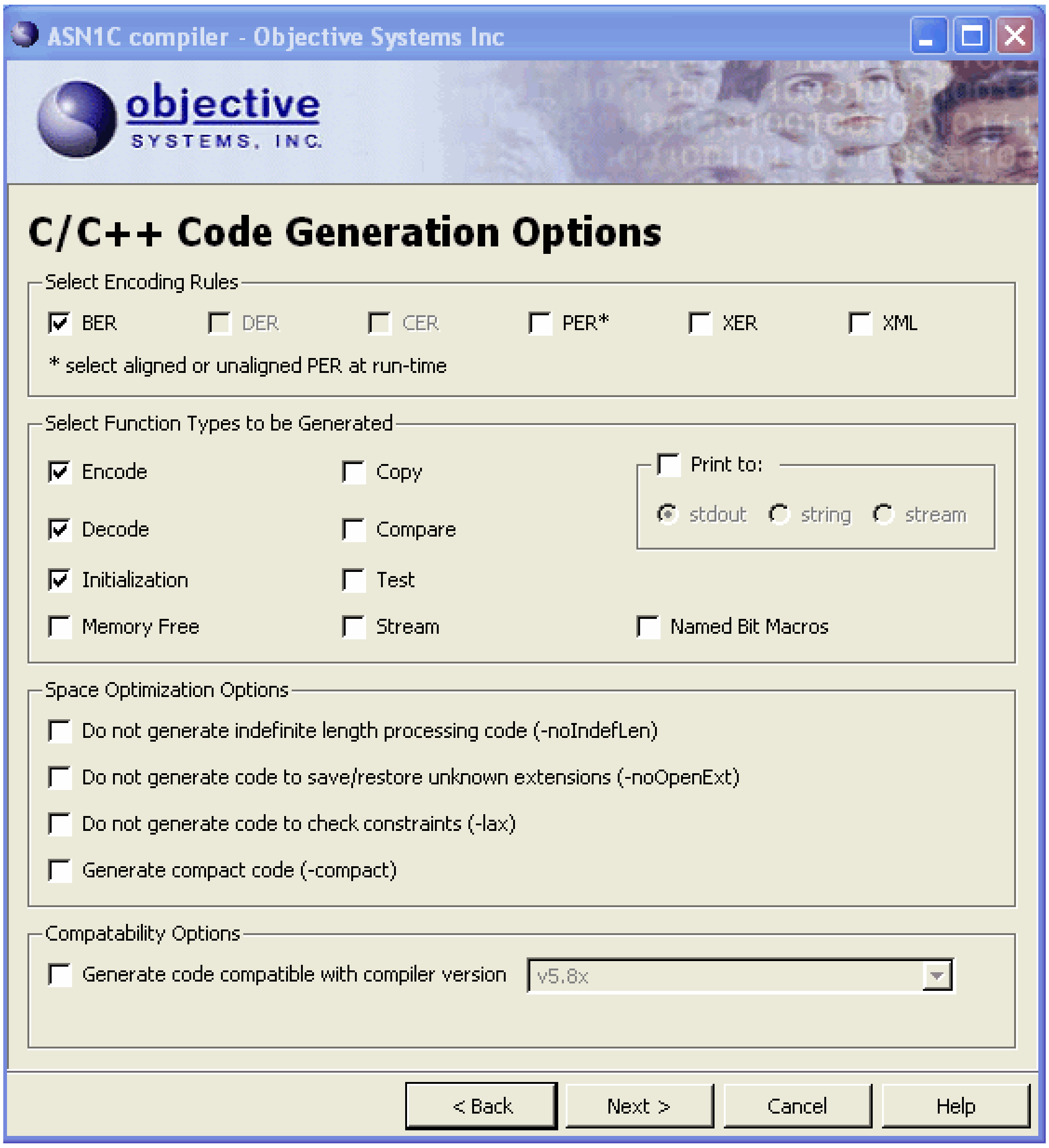
- Generate equivalent XML schema (XSD) file
- Generate ASN.1 file based on X.694 (XSD input only)

Generate code for all dependent imported type definitions

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 4

Choose Encoding Rules.

Choose from BER, DER,CER, PER, XER or XML by clicking in the option box. Select options for Function Types and Space Optimizations.



C/C++ Code Generation Options

Select Encoding Rules

[✓] BER [ ] DER [ ] CER [ ] PER* [ ] XER [ ] XML

* select aligned or unaligned PER at run-time

Select Function Types to be Generated

[✓] Encode [ ] Copy [ ] Print to: ([● stdout] string stream)

[✓] Decode [ ] Compare

[✓] Initialization [ ] Test

[ ] Memory Free [ ] Stream [ ] Named Bit Macros

Space Optimization Options

[ ] Do not generate indefinite length processing code (-noIndefLen)

[ ] Do not generate code to save/restore unknown extensions (-noOpenExt)

[ ] Do not generate code to check constraints (-lax)

[ ] Generate compact code (-compact)

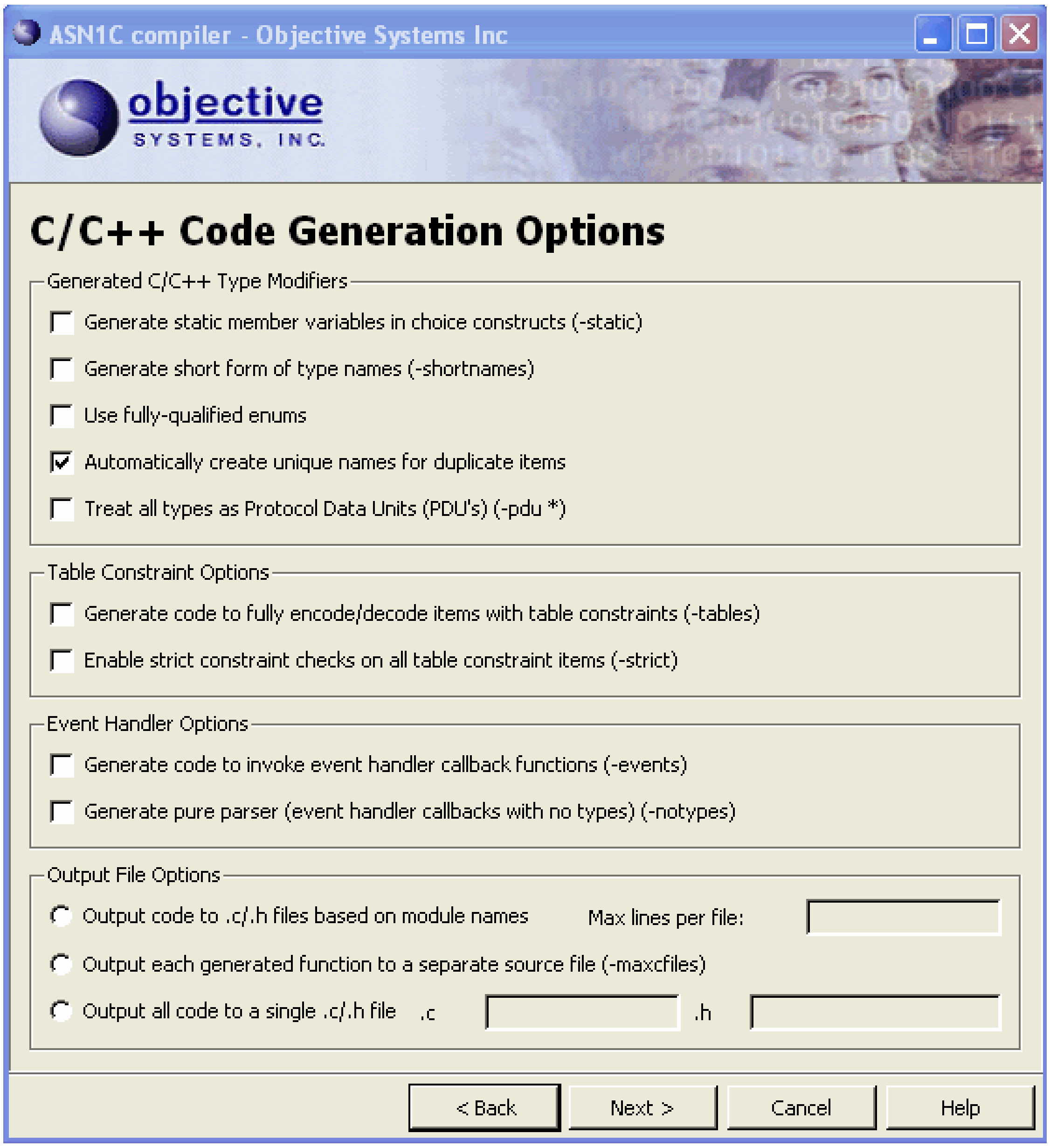
Compatability Options

[ ] Generate code compatible with compiler version v5.8x

These pages have only optional commands for the ASN1C compiler. It is not necessary to select any of these options for the most basic parsing of your ASN.1 file.

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 5

Select desired options for code generation.


ASN1C compiler - Objective Systems Inc

C/C++ Code Generation Options

Generated C/C++ Type Modifiers

- ☐ Generate static member variables in choice constructs (-static)
- ☐ Generate short form of type names (-shortnames)
- ☐ Use fully-qualified enums
- ☑ Automatically create unique names for duplicate items
- ☐ Treat all types as Protocol Data Units (PDU's) (-pdu *)

Table Constraint Options

- ☐ Generate code to fully encode/decode items with table constraints (-tables)
- ☐ Enable strict constraint checks on all table constraint items (-strict)

Event Handler Options

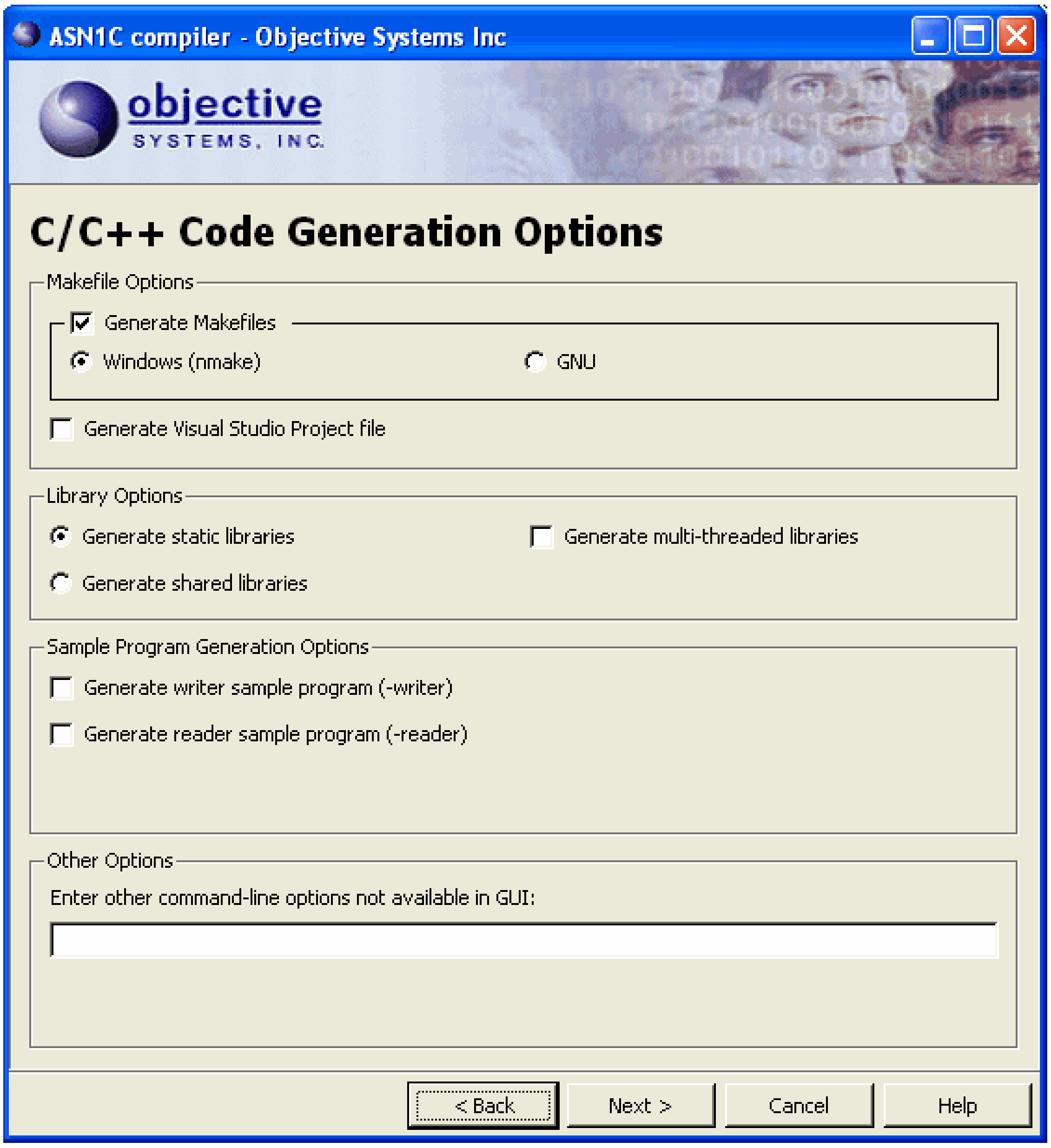
- ☐ Generate code to invoke event handler callback functions (-events)
- ☐ Generate pure parser (event handler callbacks with no types) (-notypes)

Output File Options

- ○ Output code to .c/.h files based on module names — Max lines per file:
- ○ Output each generated function to a separate source file (-maxcfiles)
- ○ Output all code to a single .c/.h file — .c — .h

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 6

Choose desired options for C, C++, code generation.


ASN1C compiler - Objective Systems Inc

C/C++ Code Generation Options

Makefile Options

[✓] Generate Makefiles

(●) Windows (nmake) ( ) GNU

[ ] Generate Visual Studio Project file

Library Options

(●) Generate static libraries [ ] Generate multi-threaded libraries

( ) Generate shared libraries

Sample Program Generation Options

[ ] Generate writer sample program (-writer)

[ ] Generate reader sample program (-reader)

Other Options

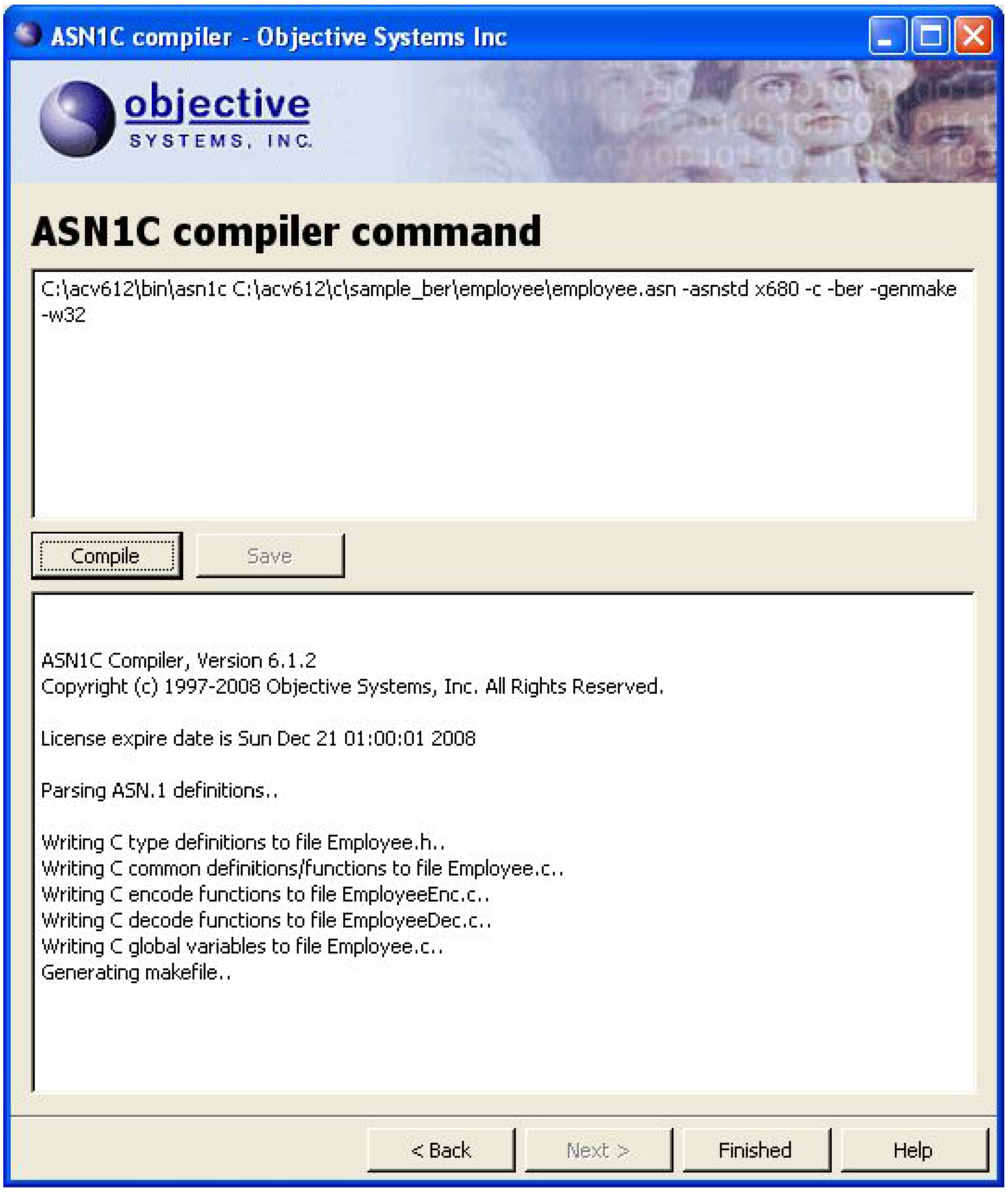
Enter other command-line options not available in GUI:

Click on next button to move to the compile screen.

Using the ASN1C Graphical User Interface (GUI) Wizard - Page 7

When you reach the Complier Command Window, select the compile button.

Note the contents of the text block below the compile button. If the file compiles without error, the ASN1C compiler will display a message stating compilation was successful. If errors were encountered, the text block will display them.



```
C:\acv612\bin\asn1c C:\acv612\c\sample_ber\employee\employee.asn -asnstd x680 -c -ber -genmake
-w32
```

```
ASN1C Compiler, Version 6.1.2
Copyright (c) 1997-2008 Objective Systems, Inc. All Rights Reserved.

License expire date is Sun Dec 21 01:00:01 2008

Parsing ASN.1 definitions..

Writing C type definitions to file Employee.h..
Writing C common definitions/functions to file Employee.c..
Writing C encode functions to file EmployeeEnc.c..
Writing C decode functions to file EmployeeDec.c..
Writing C global variables to file Employee.c..
Generating makefile..
```

Select the Finish button to close the ASN1C wizard when you are complete.

ASN1C compiler wizard help

NOTE: The documentation related to Asn1c compiler & runtime library can be found at http://www.obj-sys.com/docs/documents.shtml

ASN1C compiler input

Add ASN.1 files

The **Add ASN.1 File** button allows Abstract Syntax Notation One (ASN.1) source files to be added to the project. These source files contain the ASN.1 productions that define ASN.1 type and/or value specifications to be compiled. To add a file, press the **Add** button next to the ASN.1 file window pane and select the file from the directory view window. Multiple source files can be added by clicking the button multiple times. Files can be removed by highlighting file names in the panel and clicking **Remove**

Add Include Directories

This item is optional. It is used to specify directories that the compiler will search for ASN.1 source files for IMPORT items. Directory paths are specified in the same way as ASN.1 files above using the **Add** and **Remove** buttons next to the panel.

Add Configurations Files

This item is optional. It is used to specify the name of a file containing configuration information for the source file(s) being parsed. Configuration files can have a .cfg or .xml extension. A full discussion of the contents of a configuration file is provided in the ASN1C C/C++ or Java or C# manual. Configuration files are specified in the same way as ASN.1 files above using the **Add** and **Remove** buttons next to the panel.

top

Code generation options

Choose ASN.1 standard category

The 1997 / 2002 ASN.1 syntax is the default. If the ASN.1 source file is based on the 1990 ASN.1 standard (x.208), the 1990 ASN.1 syntax compiler should be used. The 1990 version can also parse embedded ROSE and SNMP macro definitions.

Choose Encoding/Decoding Rules

Select one of the following options (note: this is a required field):

BER	This option instructs the compiler to generate functions that implement the Basic Encoding Rules (BER) as specified in the ASN.1 X.690 standard.
DER	This option instructs the compiler to generate functions that implement the Distinguished Encoding Rules (DER) as specified in the ASN.1 X.690 standard.
CER	This option instructs the compiler to generate functions that implement the Canonical Encoding Rules (CER) as specified in the ASN.1 X.690 standard.
PER	This option instructs the compiler to generate functions that implement the Packed Encoding Rules (PER) as specified in the ASN.1 X.691 standard.
XER	This option instructs the compiler to generate functions that implement the XML Encoding Rules (XER) as specified in the ASN.1 X.693 standard.
XML	This option instructs the compiler to generate functions that implement the XML Encoding Rules (XML) as specified in the World-Wide Consortium (W3C). Related Xml Schema can be generated with Asn1c compiler -xsd option.

Choose Application Language

Select one of the following options (note: this is a required field):

C	Generate C source code.

C++	Generate C++ source code.
C#	Generated C# source code.
Java	Generate Java source code.
Xml Schema	Generate Xml Schema source code.

Note: selection of an item in each of the previous two fields is required to enable the Next button to move on to the next window.

top

Additional compiler options

All entries on this page are optional. Only select the additional code generation features you want to enable. All are off by default. Some of the following options will not be displayed for XML Schmea code generation. Addition code generation options are divided in following subsections:

Code Reduction Options

Generate compact code

This option instructs the compiler to generate more compact code at the expense of some constraint and error checking. This is an optimization option that should be used after an application is thoroughly tested. Note: this option is not available for Xml Schema code generation.

Do not generate encode functions

This option suppresses the generation of encode functions. By default, the compiler will generate both encode and decode functions. Note: this option is not available for Xml Schema code generation.

Do not generate decode functions

This option suppresses the generation of decode functions. By default, the compiler will generate both encode and decode functions. Note: this option is not available for Xml Schema code generation.

Do not generate indefinite length tests

This option instructs the compiler to omit indefinite length tests in generated decode functions. These tests result in the generation of a large amount of code. If you know that your application only uses definite length encoding, this option can result in a much smaller code base size. (Note: this option applies only to BER/DER code generation and has no meaning for other encoding rules).

Do not generate open extension elements

This option instructs the compiler to not add an open extension element(extElem1) in constructs that contain extensibility markers. The purpose of the element is to collect any unknown items in a message. If an application does not care about these unknown items, it can use this option to reduce the size of the generated code.

Do not generate code to check constraints

This option is used to tell the compiler to not generate the code to handle the Asn1 constraints. Type with constraint will be processed as type without constraints.

Options to Generate Additional Code

Generate named bit macro

This option instructs the compiler to generate macros to set, clear, or test the named bit in a bit string structure. These macros offer better performance then using the run-time functions because all calculations of mask and index values are done at compiler time. The drawback is they can result in a large amount of additional generated code.

Generate free routines

This option works only for C code generation.

This option instructs the compiler to generate a memory free function for each ASN.1 production. Normally, memory is freed within ASN1C by using the rtMemFree run-time function to free all memory at once that is held by a context. Generated free functions allow finer grained control over memory freeing by just allowing the memory held for specific objects to be freed.

Generate print routines

This option indicates print functions should be generated. Print functions are debug functions that allow the contents of generated type variables to be written to stdout. This option is optional: if not selected, print functions are not generated.

- C/C++ print functions are written to a file named "<module>Print.c" (or .cpp) where <module> is the name of the ASN.1 module.
- Java print method is generated in each of the generated Java source files.

- C# print method is generated in each of the generated C# source files.

Add trace diagnostic messages to generated code

This option is used to tell the compiler to add trace diagnostic messages to the generated code. These messages cause printf statements to be added to the generated code to print entry and exit information into the generated functions. This is a debugging option that allows encode/decode problems to be isolated to a given production processing function. Once the code is debugged, this option should not be used as it adversely affects performance. Note: this option is not available for Xml Schema code generation.

Options to Alter Generated Code

Generate strict constraints check

This option is used to tell the compiler to generate the code to handle the Asn1 constraints, which are not required practiaclly. This option will increase the size of the generated code & reduces the generated code performance. The affected constraint is Table Constraint value fields.

Generate static elements

This option has the same effect as specifying the global static configuration item. The compiler will insert static elements instead of pointer variables in generated structures.

Generate code compatible with older compiler version

This option instructs the compiler to generate code compatible with an older version of the compiler. The compiler will attempt to generate code more closely aligned with the given previous release of the compiler. is specified as x.x (for example, -compat 5.2)

Pro Version Options

Generate test routines

This option avaliable only for C and C++ code generation.

This option allows the specification of a C or C++ source (.c or .cpp) file to which generated “ test” functions will be written. “ Test” functions are used to populate an instance of a generated PDU type variable with random test data. This instance can then be used in an encode function call to test the encoder. Another advantage of these functions is that they can act as templates for writing your own population functions. The <filename> argument to this option is optional. If not specified, the functions will be written to <modulename>Test.c where <modulename> is the name of the module from the ASN.1 source file.

Generate separate file for each function

This option avaliable only for C and C++ code generation.

This option instructs the compiler to generate a separate .c file for each generated C function. In the case of C++, a separate .cpp file is generated for each control class, type, C function. This is a space optimization option – it can lead to smaller executable sizes by allowing the linker to only link in the required program module object files.

Generate print to stream routines

This option avaliable only for C and C++ code generation.

This option allows the specification of a C or C++ source (.c or .cpp) file to which generated “print-to-stream” functions will be written. “Print-to-stream” functions are similar to print functions except that the output is written to a user-provided stream instead of stdout. The stream is in the form of an output callback function that can be set within the run-time context making it possible to redirect output to any type of device. The <filename> argument to this option is optional. If not specified, the functions will be written to <modulename>Print.c where <modulename> is the name of the module from the ASN.1 source file.

Generate print to text buffer routines

This option avaliable only for C and C++ code generation.

This option indicates print functions should be generated. These are the same as print functions except the output is redirected to a user given text buffer instead of stdout. This makes it possible to display the information on a different output device (such as a Window). C/C++ print functions are written to a file named "<module>Print.c" (or .cpp) where <module> is the name of the ASN.1 module. This option is optional: if not selected, these functions are not generated.

Generate copy routines

This option avaliable only for C and C++ code generation.

This option indicates copy functions should be generated. These are functions that allow one populated data variable to be copied to another. A deep copy of all dynamic data is done using the ASN1C memory management function. This item is optional: if not selected, these functions are not generated.

Generate comparison routines

This option avaliable only for C and C++ code generation.

This option indicates comparison functions should be generated. Comparison functions are debug functions that allow the contents of generated type variables to be compared with other variables of the same type. C/C++ comparison functions are written to a file named "<module>Print.c" (or .cpp) where <module> is the name of the ASN.1 module. This option is optional: if not selected, comparison functions are not generated.

Generate makefile for project

This option avaliable only for C and C++ code generation.

This option instructs the compiler to generate a portable makefile for compiling the generated C or C++ code. The window or unix format are required to be selected. By default compiler will generate unix format makefile.

Generate BER stream-based encode/decode functions

This option avaliable only for BER encoding rule.

This option instructs the compiler to generate stream-based encoders/decoders instead of memory buffer based. This makes it possible to encode directly to or decode directly from a source or sink such as a file or socket. In the case of BER, it will also cause forward encoders to be generated which will use indefinite lengths for all constructed elements in a message.

Generate code to handle table constraints

This option indicates table constraint processing structures and functions should be generated. These add logic for handling types that use message tables for defining what is to be included in a give protocol interaction message. The added logic makes it possible to encode or decode these multi-part messages in a single step. Note: this option is not available for Xml Schema or XER rule.

Generate event handlers

Generate extra code to invoke user defined event and error handler callback methods (see the Event Handlers section of the ASN1C C/C++ or Java or C# Manual). Note: this option is not available for C and Xml Schema code generation or XER and XML encoding rule.

Generate event handlers with no type definitions (SAX parser)

This options generates event handlers and suppresses the generation of type definitions. It is pure parser functions. Note: this option is not available for C and Xml Schema code generation or XER and XML encoding rule.

Other Options

Output compiler warning messages

Output information on compiler generated warnings.

Generate listing

This will dump the source code to the standard output device as it is parsed. This can be useful for finding parse errors.

Output directory name

This option displays the name of a directory to which all of the generated files will be written. The default directory to which the generated source files will be written will be displayed in the box when the window is displayed. The output directory name can be changed to write the files to a different location. The directory name can either be typed in directly or the "..." button on right of field will allow the directory to be selected via a directory tree-view window.

top

C/C++ Code Generation Options

All of the following options are related to C or C++ code generation only. They are disabled if Java or C# or Xml Schema code generation is selected.

Following options can be used to change the generated source code filenames.

Change source file name

This option allows the specification of a C or C++ source (.c or .cpp) file to which all of the generated encode/decode functions will be written. The default name is <modulename>.c where <modulename> is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

Change header file name

This option allows the specification of a header (.h) file to which all of the generated typedefs and function prototypes will be written. The default name is <modulename>.h where <modulename>is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

Change print routine file name

This option is available when “Generate print routines” option is selected. This allows an alternate source file name to be specified to which the generated print routines will be written. If not specified, the print functions will be written to <modulename>Print.c (or .cpp) where <modulename> is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

Change print to buffer routine file name

This option is available when "Generate print to text buffer routines" option is selected. This allows an alternate source file name to be specified to which the generated print to buffer routines will be written. If not specified, the print functions will be written to <modulename>PrtToStr.c (or .cpp) where <modulename> is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

Change print to stream routine file name

This option is available when "Generate print to stream routines" option is selected. This allows an alternate source file name to be specified to which the generated print to stream routines will be written. If not specified, the print functions will be written to <modulename>PrtToStrm.c (or .cpp) where <modulename> is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

Change comparision routine file name

This option is available when "Generate comparison routines" option is selected. This allows an alternate source file name to be specified to which the generated comparison routines will be written. If not specified, the print functions will be written to <modulename>Compare.c (or .cpp) where <modulename> is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

Change copy routine file name

This option is available when one or more of the "Generate copy routines" option is selected. This allows an alternate source file name to be specified to which the generated copy routines will be written. If not specified, the print functions will be written to <modulename>Copy.c (or .cpp) where <modulename> is the name of the module from the ASN.1 source file.

Note: This option is disabled, if " Generate seperate file for each function" is selected on previous page.

top

C# Code Generation Options

All of the following options are related to C# code generation only. They are disabled if C or C++ or Java or Xml Schema code generation is selected.

Generate code in module name directory

This option indicates each module code will be generated in respective module name directory. Compiler will create a directory for module name if directory is not exist. Created directory name will be module name by replacing _ for - (i.e. ACSE-1 code will be generated in ACSE_1 directory)

Generate short names for production

This option indicates the change in the name generated by compiler for embedded type in constructive type. This option is required to handle the limit on the size of the filename. With this option, generated code filenames would be shorter than without this option.
Without this option compiler will generates embedded type with a name:
<TypeName>_<ElementName>_..._<LastElementName>
With this option compiler will generates embedded type with a name:
<TypeName>_<LaseElementName>_%d (number would be added for duplicate names)

C# NameSpace Prefix

This is a C# option for adding a prefix in front of the assigned C# namespace name. By default, the C# namespace name is set to the module name. If the namespace is embedded within a hierarchy, this option can be used to set the other directory names that must be added to allow C# to find the .cs files. This option cannot be used in conjunction with the C# NameSpace Name option.

C# NameSpace Name

This is a C# option that allows the entire C# namespacee name to be changed. Instead of the module name, the full name specified using this option will be used. This option cannot be used in conjunction with the C# NameSpace Prefix option

top

Java Code Generation Options

All of the following options are related to Java code generation only. They are disabled if C or C++ or C# or Xml Schema code generation is selected.

Generate code in module name directory

This option indicates each module code will be generated in respective module name directory. Compiler will create a directory for module name if directory is not exist. Created directory name will be module name by replacing _ for - (i.e. ACSE-1 code will be generated in ACSE_1 directory)

Generate short names for production

This option indicates the change in the name generated by compiler for embedded type in constructive type. This option is required to handle the limit on the size of the filename. With this option, generated code filenames would be shorter than without this option.
Without this option compiler will generates embedded type with a name:
<TypeName>_<ElementName>_..._<LastElementName>
With this option compiler will generates embedded type with a name:
<TypeName>_<LaseElementName>_%d (number would be added for duplicate names)

Java Package Prefix

This is a Java option for adding a prefix in front of the assigned Java package name. By default, the Java package name is set to the module name. If the package is embedded within a hierarchy, this option can be used to set the other directory names that must be added to allow Java to find the .class files. This option cannot be used in conjunction with the Java Package Name option.

Java Package Name

This is a Java option that allows the entire Java package name to be changed. Instead of the module name, the full name specified using this option will be used. This option cannot be used in conjunction with the Java Package Prefix option

top

Xml Schema Code Generation Options

All of the following options are related to Xml Schema code generation only. They are disabled if C or C++ or C# or Java code generation is selected.

Generate annotation for ASN.1 Tags

This option instructs the compiler to generate annotation (appinfo) for ASN.1 tag definition. In general, these tags have no meaning in an XSD representation of an ASN.1 type that is used to create or validate XML markup. However, if the schema definition is to be used to generate a BER or DER instance of a type, the tag information will be required. This option is optional: if not selected, tag information annotation are not generated.

Generate XER compatible Xml Schema

This option instructs the compiler to generate XER compatible Xml Schema, Which can be used with schema validators to validate the XER encoded message. This option is optional: if not selected, Xml Schema would be generated as per mapping rule found in c/c++ user manual.

Change target namespace

This option instructs the compiler to generate code with a user supplied target name space. If not specified, the compiler will use "http://www.obj-sys.com" as a target namespace for generated module. This same target name space will be used for imported module namespace.

Do not generate appinfo comments

This option instructs the compiler to not generate appinfo codes for enumerated, named integer & named bit types. If not specified, the compiler will generate appinfo code as specified in XmlSchema code generation topic in user manual.

top

ASN1C Compiler Output

This is the last page of the wizard. This page displays the ASN1C command-line command that was generated for the selected set of options. The **compile** button can now be pressed to execute the command. It is also possible to capture the command on the clipboard for future use in a makefile or script by selecting the command and pressing the "Ctrl-C" (control C) key on the keyboard.

top

Objective Systems Inc

http://www.obj-sys.com

Phone: (484) 875-9842

If you have any questions or concerns, Please email us at support@obj-sys.com

Generated Encode/Decode Function and Methods

Generated Makefile

The -genmake option causes a portable makefile to be generated to assist in the C or C++ compilation of all of the generated C or C++ source files. This makefile contains a rule to invoke ASN1C to regenerate the .c and .h files if any of the dependent ASN.1 source files are modified. It also contains rules to compile all of the C or C++ source files. Header file dependencies are generated for all the C or C++ source files.

Two basic types of makefiles are generated:

1. A GNU compatible makefile. This makefile is compatible with the GNU make utility which is suitable for compiling code on Linux and many UNIX operating systems, and
2. A Microsoft Visual Studio compatible makefile. This makefile is compatible with the Microsoft Visual Studio nmake utility.

A GNU compatible makefile is produced by default, the Microsoft compatible file is produced when the –w32 command line option is specified in addition to –genmake.

Both of these makefile types rely on definitions in the platform.mk make include file. This file contains parameters specific to different compiler and linker utilities available on different platforms. Typically, all the needs to be done to port to a different platform is to adjust the parameters in this file.

When a makefile is generated, it is assumed that the ASN1C project exists within the ASN1C installation directory tree. The generation logic tries to determine the root directory of the installation by traversing upward from the project directory in an attempt to locate the rtsrc subdirectory which is assumed to be the installation root directory. The makefile variable OSROOTDIR is then set to this value. A similar traversal is done to locate the platform.mk and xmlparser.mk files. These paths are then set in the makefile. If the project directory is located outside of the ASN1C directory tree, the user must set the OSROOTDIR environment variable to point at the ASN1C root directory in order for the makefile generation to be successful. If this is done, it is assumed that the platform.mk and xmlparser.mk files are located in this directory as well. If the compiler is unable to determine the root directory using any of the methods described above, an error will be generated and the user will need to manually edit the makefile to set the required root directory parameters and makefile include file paths.

Generated VC++ Project Files

The -vcproj option causes Microsoft Visual Studio project and workspace files to be generated that can be used to build the generated code. The files are compatible with Visual Studio version 6.0; but higher versions of Visula Studio can convert these files to the newer formats. This option can be used with the -dll option that will generate project files to compile all generated code into a DLL and -mt that will add multi-threaded compilation options to generated projects.

Because there are several different versions of Visual Studio, the -vcproj option takes an optional argument: the release year of the version of Visual Studio used. This modifies the resulting project to link against the appropriate set of libraries distributed with ASN1C. If no year is specified, the project will link against the usual c and cpp directories. If 2003 is specified, the project will us the c_vs2003 and cpp_vs2003 directories. If 2005 is specified, c_vs2005 and cpp_vs2005 will be used. Likewise, if 2008 is specified, c_vs2008 and cpp_vs2008 will be used.

Encode/Decode Function Prototypes

If BER or DER encoding is specified, a BER encode and decode function prototype is generated for each production (DER uses the same form – there are only minor differences between the two types of generated functions). These prototypes are of the following general form:

```
int asn1E_<ProdName> (OSCTXT* pctxt,
   <ProdName>* pvalue, ASN1TagType tagging);

int asn1D_<ProdName> (OSCTXT* pctxt,
   <ProdName>* pvalue, ASN1TagType tagging, int length);
```

The prototype with the asn1E_ prefix is for encoding and the one with asn1D_ is for decoding. The first parameter is a context variable used for reentrancy. This allows the encoder/decoder to keep track of what it is doing between function invocations.

The second parameter is for passing the actual data variable to be encoded or decoded. This is a pointer to a variable of the generated type.

The third parameter specifies whether implicit or explicit tagging should be used. In practically all cases, users of the generated function should set this parameter to ASN1EXPL (explicit). This tells the encoder to include an explicit tag around the encoded result. The only time this would not be used is when the encoder or decoder is making internal calls to handle implicit tagging of elements.

The final parameter (decode case only) is length. This is ignored when tagging is set to ASN1EXPL (explicit), so users can ignore it for the most part and set it to zero. In the implicit case, this specifies the number of octets to be extracted from the byte stream. This is necessary because implicit indicates no tag/length pair precedes the data; therefore it is up to the user to indicate how many bytes of data are present.

If PER encoding is specified, the format of the generated prototypes is different. The PER prototypes are of the following general form:

```
int asn1PE_<ProdName> (OSCTXT* pctxt, <ProdName>[*] value);

int asn1PD_<ProdName> (OSCTXT* pctxt, <ProdName>* pvalue);
```

In these prototypes, the prefixes are different (a ‘P’ character is added to indicate they are PER encoders/decoders), and the tagging argument variables are omitted. In the encode case, the value of the production to be encoded may be passed by value if it is a simple type (for example, BOOLEAN or INTEGER). Structured values will still be passed using a pointer argument.

If XER encoding is specified, function prototypes are generated with the following format:

```
int asn1XE_<ProdName> (OSCTXT* pctxt, <ProdName>[*] value,
                       const char* elemName,
                       const char* attributes);

int asn1XD_<ProdName> (OSCTXT* pctxt, <ProdName>* pvalue);
```

The encode function signature includes arguments for the context and value as in the other cases. It also has an element name argument (elemName) that contains the name of the element to be encoded and an attributes argument (attributes) that can be used to encode an attributes string. The decode function is generated for PDU-types only - decoding of internally referenced types is accomplished through generated SAX handler callback functions which are invoked by an XML parser.

If XML functions are generated using the -xml switch, the function prototypes are as follows:

```
int XmlEnc_<ProdName> (OSCTXT* pctxt, <ProdName> value,
   const OSUTF8CHAR* elemName, const OSUTF8CHAR* nsPrefix);

int XmlDec_<ProdName> (OSCTXT* pctxt, <ProdName>* pvalue);
```

In this case, the encode function contains an argument for XML element name (elemName) and also namespace prefix (nsPrefix).

General Procedures for Encoding and Decoding

Encoding functions and methods generated by the ASN1C compiler are designed to be similar in use across the different encoding rule types. In other words, if you have written an application to use the Basic Encoding Rules (BER) and then later decide to use the Packed Encoding Rules (PER), it should only be a simple matter of changing a few function calls to accomplish the change. Procedures for such things as populating data for encoding, accessing decoded data, and dynamic memory management are the same for all of the different encoding rules.

This section describes common procedures for encoding or decoding data that are applicable to any of the different encoding rules. Subsequent sections will then describe what will change for the different rules.

Dynamic Memory Management

The ASN1C run-time uses specialized dynamic memory functions to improve the performance of the encoder/decoder. It is imperative to understand how these functions work in order to avoid memory problems in compiled applications. ASN1C also provides the capability to plug-in a different memory management scheme at two different levels: the high level API called by the generated code and the low level API that provides the core memory managment functionality.

The ASN1C Default Memory Manager

The default ASN1C run-time memory manager uses an algorithm called the nibble-allocation algorithm. Large blocks of memory are allocated up front and then split up to provide memory for smaller allocation requests. This reduces the number of calls required to the C malloc and free functions. These functions are very expensive in terms of performance.

The large blocks of memory are tracked through the ASN.1 context block (OSCTXT) structure. For C, this means that an initialized context block is required for all memory allocations and deallocations. All allocations are done using this block as an argument to routines such as rtxMemAlloc. All memory can be released at once when a user is done with a structure containing dynamic memory items by calling rtxMemFree. Other functions are available for doing other dynamic memory operations as well. See the C/C++ Run-time Reference Manual for details on these.

High Level Memory Management API

The high-level memory management API consists of C macros and functions called in gemerated code and/or in application programs to allocate and free memory within the ASN1C run-time.

At the top level are a set of macro definitions that begin with the prefix rtxMem. These are mapped to a set of similar functions that begin with the prefix rtxMemHeap. A table showing this basic mapping is as follows:

Macro	Function	Description
rtxMemAlloc	rtxMemHeapAlloc	Allocate memory
rtxMemAllocZ	rtxMemHeapAllocZ	Allocate and zero memory

Macro	Function	Description
rtxMemRealloc	rtxMemHeapRealloc	Reallocate memory
rtxMemFree	rtxMemHeapFreeAll	Free all memory in context
rtxMemFreePtr	rtxMemHeapFreePtr	Free a specific memory block

See the ASN1C C/C++ Common Runtime Reference Manual for further details on these functions and macros.

It is possible to replace the high-level memory allocation functions with functions that implement a custom memory management scheme. This is done by implementing some (or all) of the C rtxMemHeap functions defined in the following interface (note: a default implementation is shown that replaces the ASN1C memory manager with direct calls to the standard C run-time memory management functions):

```
#include <stdlib.h>
#include "rtxMemory.h"

/* Create a memory heap */
int rtxMemHeapCreate (void** ppvMemHeap) {
   return 0;
}

/* Allocate memory */
void* rtxMemHeapAlloc (void** ppvMemHeap, int nbytes) {
   return malloc (nbytes);
}

/* Allocate and zero memory */
void* rtxMemHeapAllocZ (void** ppvMemHeap, int nbytes) {
   void* ptr = malloc (nbytes);
   if (0 != ptr) memset (ptr, 0, nbytes);
   return ptr;
}

/* Free memory pointer */
void rtxMemHeapFreePtr (void** ppvMemHeap, void* mem_p) {
   free (mem_p);
}
```

```
/* Reallocate memory */
void* rtxMemHeapRealloc (void** ppvMemHeap, void* mem_p, int nbytes_) {
   return realloc (mem_p, nbytes_);
}

/* Clears heap memory (frees all memory, reset all heap's variables) */
void rtxMemHeapFreeAll (void** ppvMemHeap) {
   /* should remove all allocated memory. there is no analog in standard memory
      management. */
}

/* Frees all memory and heap structure as well (if was allocated) */
void rtxMemHeapRelease (void** ppvMemHeap) {
   /* should free all memory allocated + free memory heap object if exists */
}
```

In most cases it is only necessary to implement the following functions: rtxMemHeapAlloc, rtxMemHeapAllocZ, rtxMemHeapFreePtr and rtxMemHeapRealloc. Note that there is no analog in standard memory management for ASN1C's rtxMemFree macro (i.e. the rtxMemHeapFreeAll function). A user would be responsible for freeing all items in a generated ASN1C structure individually if standard memory management is used.

The rtxMemHeapCreate and rtxMemHeapRelease functions are specialized functions used when a special heap is to be used for allocation (for example, a static block within an embedded system). In this case, rtxMemHeapCreate must set the ppvMemHeap argument to point at the block of memory to be used. This will then be passed in to all of the other memory management functions for their use through the OSCTXT structure.
The rtxMemHeapRelease function can then be used to dispose of this memory when it is no longer needed.

To add these definitions to an application program, compile the C source file (it can have any name) and link the resulting object file (.OBJ or .O) in with the application.

Built-in Compact Memory Management

A built-in version of the simple memory management API described above (i.e with direct calls to malloc, free, etc.) is available for users who have the source code version of the run-time. The only difference in this API with what is described above is that tracking of allocated memory is done through the context. This makes it possible to provide an implementation of the rtxMemHeapFreeAll function as described above. This memory management scheme is slower than the default manager (i.e. nibble-based), but has a smaller code footprint.

This form of memory management is enabled by defining the _MEMCOMPACT C compile time setting. This can be done by either adding -D_MEMCOMPACT to the C compiler command-line arguments, or by uncommenting this item at the beginning of the rtxMemory.h header file:

```
/*
 * Uncomment this definition before building the C or C++ run-time
 * libraries to enable compact memory management.  This will have a
 * smaller code footprint than the standard memory management; however,
 * the performance may not be as good.
 */
/*#define _MEMCOMPACT*/
```

Low Level Memory Management API

It is possible to replace the core memory management functions used by the ASN1C run-time memory manager. This has the advantage of preserving the existing management scheme but with the use of different core functions. Using different core functions may be necessary on some systems that do not have the standard C run-time functions malloc, free, and realloc, or when extra functionality is desired.

To replace the core functions, the following run-time library function would be used:

```
   void rtxMemSetAllocFuncs (OSMallocFunc malloc_func,
      OSReallocFunc realloc_func, OSFreeFunc free_func);
```

The malloc, realloc, and free functions must have the same prototype as the standard C functions. Some systems do not have a realloc-like function. In this case, realloc_func may be set to NULL. This will cause the malloc_func/free_func pair to be used to do reallocations.

This function must be called before the context initialization function (rtInitContext) because context initialization requires low level memory management facilities be in place in order to do its work.

Note that this function makes use of static global memory to hold the function definitions. This type of memory is not available in all run-time environments (most notably Symbian). In this case, an alternative function is provided for setting the memory functions. This function is rtxInitContextExt which must be called in place of the standard context initialization function (rtInitContext). In this case, there is a bit more work required to initialize a context because the ASN.1 subcontext must be manually initialized. This is an example of the code required to do this:

```
int stat = rtxInitContextExt (pctxt, malloc_func, realloc_func, free_func);
if (0 == stat) {
   /* Add ASN.1 error codes to global table */
   rtErrASN1Init ();

   /* Init ASN.1 info block */
   stat = rtCtxtInitASN1Info (pctxt);
}
```

Memory management can also be tuned by setting the default memory heap block size. The way memory management works is that a large block of memory is allocated up front on the first memory management call. This block is then subdivided on subsequent calls until the memory is used up. A new block is then started. The default value is 4K (4096) bytes. The value can be set lower for space constrained systems and higher to improve performance in systems that have sufficient memory resources. To set the block size, the following run-time function should be used:

```
void rtxMemSetDefBlkSize (OSUINT32 blkSize);
```

This function must be called prior to context initialization.

C++ Memory Management

In the case of C++, the ownership of memory is handled by the control class and message buffer objects. These classes share a context structure and use reference counting to manage the allocation and release of the context block. When a message buffer object is created, a context block structure is created as well. When this object is then passed into a control class constructor, its reference count is incremented. Then when either the control class object or message buffer object are deleted or go out of scope, the count is decremented. When the count goes to zero (i.e. when both the message buffer object and control class object go away) the context structure is released.

What this means to the user is that a control class or message buffer object must be kept in scope when using a data structure associated with that class. A common mistake is to try and pass a data variable out of a method and use it after the control and message buffer objects go out of scope. For example, consider the following code fragment:

```
ASN1T_<type>* func2 () {
      ASN1T_<type>* p = new ASN1T_<type> ();
      ASN1BERDecodeBuffer decbuf;
      ASN1C_<type> cc (decbuf, *p);

      cc.Decode();

      // After return, cc and decbuf go out of scope; therefore
      // all memory allocated within struct p is released..

      return p;
      }

   void func1 () {
      ASN1T_<type>* pType = func2 ();

      // pType is not usable at this point because dynamic memory
      // has been released..
   }
```

As can be seen from this example, once func2 exits, all memory that was allocated by the decode function will be released. Therefore, any items that require dynamic memory within the data variable will be in an undefined state.

An exception to this rule occurs when the type of the message being decoded is a Protocol Data Unit (PDU). These are the main message types in a specification. The ASN1C compiler designates types that are not referenced by any other types as PDU types. This behavior can be overridden by using the -pdu command line argument or <isPDU> configuration file element.

The significance of PDU types is that generated classes for these types are derived from the ASN1TPDU base class. This class holds a reference to a context object. The context object is set by Decode and copy methods. Thus, even if control class and message buffer objects go out of scope, the memory will not be freed until the destructor of an ASN1TPDU inherited class is called. The example above will work correctly without any modifications in this case.

Another way to keep data is to make a copy of the decoded object before it goes out of scope. A method called newCopy is also generated in the control class for these types which can be used to create a copy of the decoded object. This copy of the object will persist after the control class and message buffer objects are deleted. The returned object can be deleted using the standard C++ delete operator when it is no longer needed.

Returning to the example above, it can be made to work if the type being decoded is a PDU type by doing the following:

```
ASN1T_<type>* func2 () {
   ASN1T_<type> msgdata;
   ASN1BERDecodeBuffer decbuf;
   ASN1C_<type> cc (decbuf, msgdata);

   cc.Decode();

   // Use newCopy to return a copy of the decoded item..

   return cc.newCopy();
}
```

Populating Generated Structure Variables for Encoding

Prior to calling a compiler generated encode function, a variable of the type generated by the compiler must be populated. This is normally a straightforward procedure – just plug in the values to be encoded into the defined fields. However, things get more complicated when more complex, constructed structures are involved. These structures frequently contain pointer types which means memory management issues must be dealt with.

There are three alternatives for managing memory for these types:

1. Allocate the variables on the stack and plug the address of the variables into the pointer fields,
2. Use the standard malloc and free C functions or new and delete C++ operators to allocate memory to hold the data, and
3. Use the rxtMemAlloc and rtxMemFree run-time library functions or their associated macros.

Allocating the variables on the stack is an easy way to get temporary memory and have it released when it is no longer being used. But one has to be careful when using additional functions to populate these types of variables. A common mistake is the storage of the addresses of automatic variables in the pointer fields of a passed-in structure. An example of this error is as follows (assume A, B, and C are other structured types):

```
typedef struct {
   A* a;
   B* b;
   C* c;
} Parent;

void fillParent (Parent* parent)
{
   A aa;
   B bb;
   C cc;

   /* logic to populate aa, bb, and cc */
   ...

   parent->a = &aa;
   parent->b = &bb;
   parent->c = &cc;
}

main ()
{
   Parent parent;

   fillParent (&parent);

   encodeParent (&parent); /* error! pointers in parent
                              reference memory that is
                              out of scope */
   ...
}
```

In this example, the automatic variables aa, bb, and cc go out of scope when the fillParent function exits. Yet the parent structure is still holding pointers to the now out of scope variables (this type of error is commonly known as “dangling pointers”).

Using the second technique (i.e., using C malloc and free) can solve this problem. In this case, the memory for each of the elements can be safely freed after the encode function is called. But the downside is that a free call must be made for each corresponding malloc call. For complex structures, remembering to do this can be difficult thus leading to problems with memory leaks.

The third technique uses the compiler run-time library memory management functions to allocate and free the memory. The main advantage of this technique as opposed to using C malloc and free is that all allocated memory can be freed with a single rtxMemFree call. The rtxMemAlloc macro can be used to allocate memory in much the same way as the C malloc function with the only difference being that a pointer to an initialized OSCTXT structure is passed in addition to the number of bytes to allocate. All allocated memory is tracked within the context structure so that when the rtxMemFree function is called, all memory is released at once.

Accessing Encoded Message Components

After a message has been encoded, the user must obtain the start address and length of the message in order to do further operations with it. Before a message can be encoded, the user must describe the buffer the message is to be encoded into by specifying a message buffer start address and size. There are three different types of message buffers that can be described:

1. **static:** this is a fixed-size byte array into which the message is encoded
2. **dynamic:** in this case, the encoder manages the allocation of memory to hold the encoded message
3. **stream:** in this case, the encoder writes the encoded data directly to an output stream

The static buffer case is generally the better performing case because no dynamic memory allocations are required. However, the user must know in advance the amount of memory that will be required to hold an encoded message. There is no fixed formula to determine this number. ASN.1 encoding involves the possible additions of tags and lengths and other decorations to the provided data that will increase the size beyond the initial size of the populated data structures. The way to find out is either by trial-and-error (an error will be signaled if the provided buffer is not large enough) or by using a very large buffer in comparison to the size of the data.

In the dynamic case, the buffer description passed into the encoder is a null buffer pointer and zero size. This tells the encoder that it is to allocate memory for the message. It does this by allocating an initial amount of memory and when this is used up, it expands the buffer by reallocating. This can be an expensive operation in terms of performance – especially if a large number of reallocations are required. For this reason, run-time helper functions are provided that allow the user to control the size increment of buffer expansions. See the C/C++ Run-Time Library Reference Manual for a description of these functions.

In either case, after a message is encoded, it is necessary to get the start address and length of the message. Even in the static buffer case, the message start address may be different then the buffer start address (see the section on encoding BER messages). For this reason, each set of encoding rules has a run-time C function for getting the message start address and length. See the C/C++ Run-Time Library Reference Manual for a description of these functions. The C++ message buffer classes contain the getMsgPtr, getMsgCopy , and getMsgLength methods for this purpose.

A **stream** message buffer can be used for BER encoding. This type of buffer is used when the -stream option was used to generate the encode functions. See the section on BER stream encoding for a complete description on how to set up an output stream to receive encoded data.

2-4. Sample BER Build 및 디버깅

For example, we will assume that you installed ASN1C for C/C++. To test the BER C++ encode/decode capabilities, do the following:

1. Change directory (cd) to the **.\cpp\sample_ber\employee** subdirectory. Execute the **nmake** command to build the writer and reader sample programs. **nmake** is utility program that comes with Visual C++. It may also be necessary to execute a Microsoft batch file named **VCVARS32.BAT** to set the path information so that the **nmake** utility can be found. (Important note: this assumes you are using Microsoft Visual C++ on your PC. Some PC specific include and library directories in the makefile may need to be changed to get the samples to work on your system. See the README.txt file for further details).

2. Execute the **writer.exe** program to encode the sample record. The results of the encoding will be dumped to the screen and saved in a file called *message.dat*.

3. Execute the **reader.exe** program to read and decode the contents of the *message.dat* file. This program will read the encoded record into memory, decode it, and then print the contents of the generated structure variable to standard output.

2-4-1. Sample PER Build 및 디버깅

1. Change directory (cd) to the **.\sample_per\employee** subdirectory. Execute the **nmake** command to build the writer and reader sample programs.

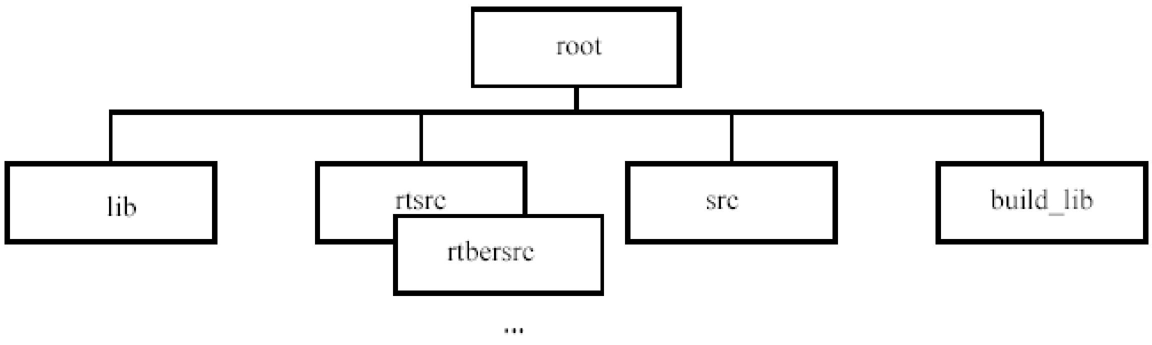
2. Execute the **writer.exe** program to encode the sample record. The **–a** switch can be used to encode a record using aligned PER. The **–u** switch encodes a record using unaligned PER. The results of the encoding will be dumped to the screen and saved in a file called *message.dat*.

3. Execute the **reader.exe** program to read and decode the contents of the *message.dat* file. The **–a** or **–u** switch must be the same as that specified when the writer program was executed. This program will read the encoded record into memory, decode it, and then print the contents of the generated structure variable to standard output. This test can be repeated for XER as well by going to the **sample_xer\employee** subdirectory and repeating the above sequence of steps.

2-4-2. 다른 Platforms 에 Run-time Code 포팅

모든 platform 사양 내용은 소프트웨어가 설치된 루트 디렉토리의 platform.mk 에 있습니다.
다른 platform 에 Run-time code 를 포팅하는 방법은 아래와 같습니다.

1. Create a directory tree containing a root directory (the name does not matter) and lib, src, rt*src, and build_lib subdirectories (note: in these definitions, * is a wildcard character indicating there are multiple directories matching this pattern). The tree should be as follows:



2. Copy the files ending in extension ".mk" from the root directory of the installation to the root directory of the target platform (note: if transferring from DOS to UNIX or vice-versa, FTP the files in ASCII mode to ensure lines are terminated properly).

3. Copy all files from the src and the different rt*src subdirectories from the installation to the src and rt*src directories on the target platform (note: if transferring from DOS to UNIX or vice-versa, FTP the files in ASCII mode to ensure lines are terminated properly).

4. Copy the makefile from the build_lib subdirectory of the installation to the build_lib subdirectory on the target platform (note: if transferring from DOS to UNIX or vice-versa, FTP the files in ASCII mode to ensure lines are terminated properly).

5. Edit the *platform.mk* file in the root subdirectory and modify the compilation parameters to fit those of the compiler of the target system. In general, the following parameters will need to be adjusted:

   a. CC C compiler executable name

b. CCC C++ compiler executable name
c. CFLAGS_ Flags that should be specified on the C or C++ command line

The *platform.w32* and *platform.gnu* files in the root directory of the installation are sample files for Windows 32 (Visual C++) and GNU compilers respectively. Either of these can be renamed to *platform.mk* for building in either of these environments.

6. Invoke the makefile in the build_lib subdirectory.

모든 파라메타 설정이 정확하다면, lib 서브디렉토리에 바이너리 라이브러리가 만들어질 것 입니다.

Library Files	Description
asn1rt_a.lib *asn1ber_a.lib* *asn1per_a.lib* *asn1xer_a.lib* *asn1xml_a.lib*	Static single-threaded libraries. These are built without -MT (multithreading) and -MD (dynamic link libraries) options. These are not thread-safe. However, they provide the smallest footprint of the different libraries.
asn1rt.lib *asn1ber.lib* *asn1per.lib* *asn1xer.lib* *asn1xml.lib*	DLL libraries. These are used to link against the DLL versions of the run-time libraries (*asn1rt.dll*, etc.)
asn1rtmt_a.lib *asn1bermt_a.lib* *asn1permt_a.lib* *asn1xermt_a.lib* *asn1xmlmt_a.lib*	Static multi-threaded libraries. These libraries were built with the -MT option. They should be used if your application contains threads and you wish to link with the static libraries (note: the DLLs are also thread-safe).
asn1rtmd_a.lib *asn1bermd_a.lib* *asn1permd_a.lib* *asn1xermd_a.lib* *asn1xmlmd_a.lib*	DLL-ready multi-threaded libraries. These libraries were built with the -MD option. They allow linking additional object modules in with the ASN1C

2-5. REFERENCE

Changing form ASN.1: 1998 to ASN.1:2002

http://www.itu.int/ITU-T/studygroups/com17/changing-ASN/

ASN.1 Module Database

http://www.itu.int/ITU-T/asn1/database/index.html

Search the ITU Web site

http://www.itu.int/search/index.html

Objective System Inc ASN1C/CPP Products

http://www.obj-sys.com/

Objective System Inc Download ASN1C Evaluations Copy

http://www.obj-sys.com/downloads.shtml

Objective System Inc AS1C Documentation

http://www.obj-sys.com/support.php)

CURRENT DOCUMENTATION **FAQ**

ASN1C Version 6.5	Release Notes	Manuals	Change Log

PREVIOUS RELEASES

ASN1C Version 6.4	Release Notes	Manuals	Change Log
ASN1C Version 6.3	Release Notes	Manuals	Change Log
ASN1C Version 6.2	Release Notes	Manuals	Change Log
ASN1C Version 6.1	Release Notes	Manuals	Change Log
ASN1C Version 6.0	Release Notes	Manuals	Change Log
ASN1C Version 5.8	Release Notes	Manuals	Change Log

TUTORIALS

Tutorial providing an introduction to the basic concepts of ASN.1

Getting Started with the ASN1C Graphical User Interface (GUI) Wizard.

Getting Started with the ASN1C Java command line.

Getting Started with the ASN1C C / C++ command line.

Using the existing Visual Studio project files.

WHITEPAPERS

ASN1C C/C++ Code Generation for 3GPP and LTE

ASN1C Mapping of ASN.1 Syntax to XML Schema

ASN1C Support for Information Objects and Parameterized Types

SAX-like Event Handlers for the Generic Parsing of ASN.1 Encoded Data

ASN1VE

CURRENT DOCUMENTATION

ASN1VE Version 2.2	Release Notes	Manuals	Change Log

PREVIOUS RELEASES

ASN1VE Version 2.1	Release Notes	Manuals	Change Log
ASN1VE Version 2.0	Release Notes	Manuals	Change Log
ASN1VE Version 1.7	Release Notes	Manuals	Change Log

ASN2XML

CURRENT DOCUMENTATION

ASN2XML Version 2.2	Release Notes	Manuals	Change Log

PREVIOUS RELEASES

ASN2XML Version 2.1	Release Notes	Manuals	Change Log

ASN2CSV

CURRENT DOCUMENTATION

ASN2CSV Version 2.1	Release Notes	PDF | HTML	Change Log

CSTADLL

CURRENT DOCUMENTATION

CSTADLL Version 1.2	Release Notes	PDF | HTML	Change Log
CSTADLL Version 1.1	Release Notes	PDF | HTML	Change Log
CSTADLL Version 1.0	Release Notes	PDF | HTML	Change Log

XBinder

CURRENT DOCUMENTATION

XBinder Version 2.2	Release Notes	Manuals	Change Log

PREVIOUS RELEASES

XBinder Version 2.1	Release Notes	Manuals	Change Log
XBinder Version 2.0	Release Notes	Manuals	Change Log
XBinder Version 1.5	Release Notes	Manuals	Change Log
XBinder Version 1.4	Release Notes	Manuals	Change Log

TUTORIALS

Introduction to XBinder.

RELATED INFORMATION

[CONFORMANCE TESTS] W3C XML Schema Patterns for Databinding Test Results

[COMPARISONS] Comparing XBinder to Non-XBinder Code

License Servers

RLM license server on Windows

RLM license server on UNIX

objsys license server on Windows

objsys license server on UNIX

General Articles and Tutorials

[PRESENTATION] W3C Efficient XML Interchange (EXI)

[WHITEPAPER] Use of ASN.1 Encoding Rules for Binary XML

[CONFORMANCE TESTS] W3C XML Schema Patterns for Databinding Test Results

General Articles and Tutorials

[PRESENTATION] W3C Efficient XML Interchange (EXI)

[WHITEPAPER] Use of ASN.1 Encoding Rules for Binary XML

THE END OF DOCUMENT