ASN.1 a kódování BER & DER, UTF-8

Libor Dostálek

Libor.Dostalek@prf.jcu.cz

ASN.1

- Deklarace proměnných v programovacích jazycích

Typy:

- Jednoduchý („vestavěný“) typ, který již dále není strukturován. Např. INTEGER, OCTET STRING, PrintableString, OBJECT IDENTIFIER atd.
- Strukturovaný typ, který se skládá z jednotlivých položek. Každá položka struktury je nějakým typem. Jednotlivé prvky struktury zapisujeme do složených závorek { } a oddělujeme vzájemně čárkou.
- Odvozený typ, který je odvozen od ostatních typů.
- Jiné typy – jedná se zejména o typy CHOICE (výběr) a ANY (vše).

Příklad:

Version ::= INTEGER

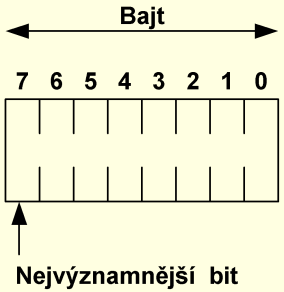
Rodiny kódování

- BER
  - DER
  - CER

- PER - (*Packet encoding rules*). BER předpokládá, že kóduje zpráva, tj. data, která mají definovaný začátek i konec. To v případě takového real audia neplatí, tak se jedná o tok paketů u kterého dokonce nevadí, když se tu a tam některý z nich ztratí. Pro kódování toku paketů je určeno právě kódování PER.

- XER (*XML encoding rules*). A tady pozor! XER specifikuje tzv. kanonický tvar XML. Mnohdy lze totiž do XML zprávy bez potíží vložit mezery, nové řádky apod. V případě elektronického podpisu (dle standardu XMLSignature) může ale nastat potíž. Taková jedna vložená/vypuštěná mezera může změnit otisk dokumentu čehož důsledkem je selhání při ověřování podpisu. Řešením je právě kanonický tvar XML zprávy. Kanonický tvar je jedinečným zápisem XML zprávy. Před podpisem (XMLSignature) právě do něj převedeme zprávu. Pak se do zprávy klidně mohou vkládat ty měkké mezery apod., ale před ověřováním podpisu opět převedeme zprávu zpět do kanonického tvaru a měli bychom dospět ke stejnému otisku.

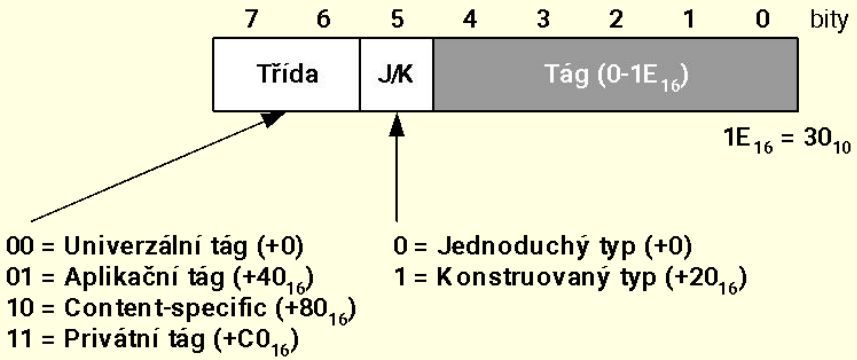
BER kódování



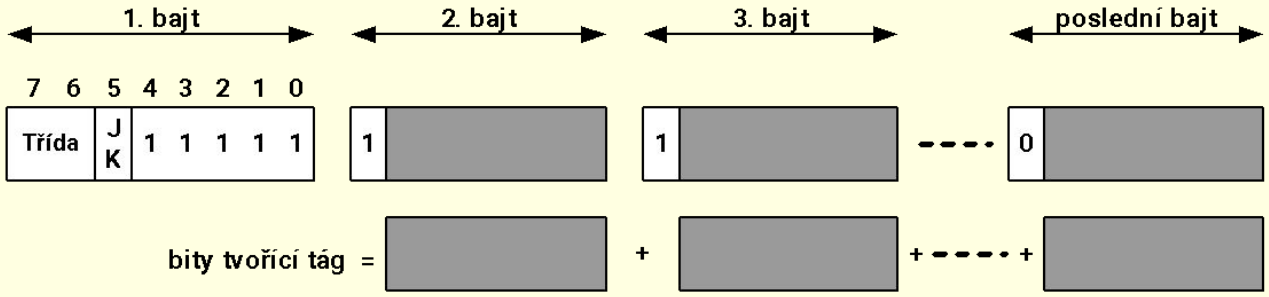
Typ dat	Délka dat	Data

Pole typ dat

Pole typ dat pro tágy menší než 31
(vejde se do jednoho bajtu)



Pole typ dat pro tágy větší než 30
(skládá se z váce bajtů)

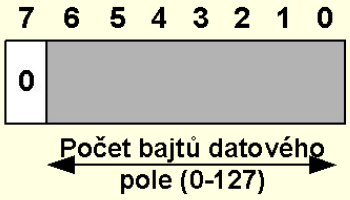


Univerzální typy

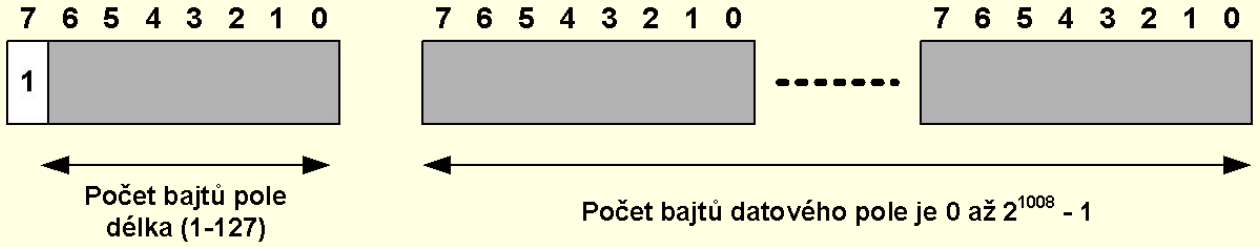
Typ	Tág desítkově	šestnáctkově	Význam
END OF CONT. (EOC)	0	0	Konec pole dat v kódování BER pro případ, že data jsou nedefinované délky.
BOOLEAN	1	1	Nabývá hodnot TRUE a FALSE.
INTEGER	2	2	Celá kladná i záporná čísla a nula.
BIT STRING	3	3	Řetězec bitů – může být zadán binárně nebo hexadecimálně.
OCTET STRING	4	4	Řetězec bajtů (blíže nespecifikovaný).
NULL	5	5	Prázdný typ.
OBJECT IDENT.	6	6	Identifikátor objektu
REAL	9	6	{ Mantisa,Base,Exponent} = M x BE
ENUMERATED	10	A	Výčet hodnot (nemusí být uspořádaný).
UTF8String	12	C	Viz odstavec 14.3
SEQUENCE a SEQUENCE OF	16	10	Uspořádaná posloupnost prvků. SEQUENCE OF může být i prázdná.
SET a SET OF	17	11	Množina prvků (neuspořádaná). SET OF může být i prázdná.
NumericString	18	12	0-9 amezera
PrintableString	19	13	0-9 a-z A-Z mezera ‘ ( ) + , – . / : = ?
T61String	20	14	TeleText podle doporučení T.61.
VideotexString	21	15	
IA5String	22	16	ASCII, tj. kromě znaků PrintableString obsahuje: SPACE DELETE a
UTCTime	23	17	Čas
GeneralisedTime	24	18	Čas
GraphicString	25	19	Grafické znaky
VisibleString	26	1A	Sada 6 a SPACE
GeneralString	27	1B	Znaky + grafické znaky + DELETE
UniversalString	28	1C	UCS-4
BMPString	30	1E	UCS-2

Pole délka dat

Krátká forma pole délka
(jeden bajt)



Dlouhá forma pole délka
(více bajtů)



Příklad

V kódování BER vyjádřete jednu položku z kartotéky zaměstnanců firmy kapitalista.cz. Tato firma má každého zaměstnance popsaného na kartě skládající se ze čtyř položek:

- **příjmení IA5String**
- **jméno IA5String**
- **pohlaví BOOLEAN**
- **známosti BOOLEAN**

Příklad - pokračování

V ASN.1 se v praxi zápis Typu Zamestnanec píše elegantněji:

```
Zamestnanec ::= SEQUENCE {
        prijmeni    Prijmeni,
        jmeno           Jmeno,
        pohlavi         Pohlavi,
        znamosti        Znamosti
}
Kde
        Prijmeni    ::=     IA5string,
        Jmeno       ::=     IA5string,
        Pohlavi             ::=     BOOLEAN,
        Znamosti    ::=     BOOLEAN}
```

Místo typu SEQUENCE by se asi měl použít typ SEQUENCE OF, protože kartotéka může být i prázdná (když všechny zaměstnance propustíme).

Příklad - dokončení

Takže konkrétní kartu v kódování BER pro zaměstnance:
Bobek Bob TRUE (Pohlavi) FALSE (Znamosti) bychom v BER konstruovali zevnitř (od jednoduchých typů):

- Bobek bude 16 05 42 6F 62 65 6B (16 je tág pro IA5STRING; řetězec je dlouhý 05 znaků)
- Bob bude 16 03 42 6F 62
- TRUE bude 01 01 FF
- FALSE bude 01 01 00

Celkově je vnitřek 16 05 42 6F 62 65 6B 16 03 42 6F 62 01 01 FF 01 01 00, tj. je dlouhý 18 bajtů, tj. šestnáctkově 12.

Nyní specifikujeme posloupnost (vnějšek). U SEQUENCE už víme, že má hodnotu šestnáctkově 30 (10+20) a je dlouhá 1216, takže výsledek:
30 12 16 05 42 6F 62 65 6B 16 03 42 6F 62 01 01 FF 01 01 00

Program dumpasn1

- Praktická ukázka

Prázdný typ

- Prázdný typ se kóduje opět dle schématu: tág (05), délka (nula) a hodnota (ta je ale prázdná). Výsledkem je pak:

  05 00 (jediné přípustné v DER)

  05 81 00

  atd.

BOOLEAN

- Pravda se kóduje: 01 01 FF
- Nepravda se kóduje: 01 01 00

INTEGER

- Typ INTEGER se kóduje binárně tak, jak je to běžné, tj. nejvýznamnější bit nastavený na 1 signalizuje záporné číslo.

Příklady kódování celých čísel:

Hodnota	***BER-kódování***
0	02 01 00
$127_{10} = 7F_{16}$	02 01 7F
$128_{10} = 80_{16}$	02 02 00 80
$256_{10} = 100_{16}$	02 02 01 00
-128_{10} ($100_{16}-80_{16} = 80_{16}$)	02 01 80
-129_{10} ($1000_{16}-81_{16}= FF7F_{16}$)	02 02 FF 7F

Výčet

- Pomocí typu INTEGER lze vytvořit výčet tak, že se jednotlivé hodnoty pojmenují identifikátory. V kulatých závorkách za identifikátorem hodnoty je pak uvedena pojmenovaná hodnota.

- Obecně: *INTEGER [{ identifikátor-1 (hodnota-1) ... identifikátor-n (hodnota-n)}]*

Příklad 14.8:
Barva ::= INTEGER { cervena(1) modra(2) bila (3)}

Pak jeden výskyt může být např.:
0A 01 02

Což můžeme vypsat programem dumpasn1:
ENUMERATED 2

SEQUENCE, SEQUENCE OF, SET a SET OF

- Jedná se o strukturované typy (k tágu se připočítává 20_{16}).
- Slovo OF v obou případech vyjadřuje, že posloupnost (resp. množina) může být i prázdná.
- V posloupnosti (SEQUENCE) záleží na pořadí prvků, v množině (SET) nikoliv.
- Syntaxe SEQUENCE o n prvcích je:

  SEQUENCE {
  [identifikátor-1] Typ-1 [{ OPTIONAL | DEFAULT hodnota-1}],
  ...
  [identifikátor-n] Typ-n [{ OPTIONAL | DEFAULT hodnota-n}] }
- Syntaxe SEQUENCE OF, SET a SET OF je obdobná, pouze s tím rozdílem, že slovo SEQUENCE je nahrazeno slovy SEQUENCE OF, resp. SET, resp. SET OF.
- Slovo OPTIONAL vyjadřuje, že prvek je volitelný (nepovinný). Slovo DEFAULT následované hodnotou pak určuje, že v případě, kdy prvek není uveden, se použije tato hodnota.
- Pro DER kódování musí být splněny dvě podmínky:
- V případě, že prvek je nepovinný (OPTIONAL) a nabývá implicitní hodnoty (DEFAULT), se neuvádí, tj. nekóduje se do DER.
- V případě množiny (SET) je pravdou, že nezáleží na pořadí, ale do kódování DER se jednotlivé prvky množiny kódují sestupně setříděny podle hodnoty tágu.

Čas

- Používáme dva typy pro čas: UTCTime (tág 23) a GeneralisedTime (tág 24). Oba plníme časem UTC (viz odstavec 20.1). Rozdíl mez těmito dvěma typy je zejména v tom, že UTCTime má pouze **dvě** cifry pro rok a GeneralisedTime má **čtyři** cifry pro rok, tj. UTCTime by se neměl používat pro datum pozdější než roku 2049.

- Pokud je na konci znak „Z“, pak se jedná o Greenwichský čas. Pokud na konci je znaménko následované čtyřcifernou číslicí, pak se jedná o místní čas a koncovka vyjadřuje jeho posunutí od greenwichského času. Pokud na konci není znak „Z“ ani není uvedená koncovka, pak se jedná o místní čas.

- Čas se vyjadřuje v jednom z následujících tvarů (GeneralisedTime má rok ve tvaru RRRR):

  RRMMDDhhmm …. místní čas
  RRMMDDhhmmZ …. greenwichský čas
  RRMMDDhhmm+hhmm …. místní čas na východ od Greenwiche
  RRMMDDhhmm-hhmm …. místní čas na západ od Greenwiche
  RRMMDDhhmmssZ
  RRMMDDhhmmss+hhmm
  RRMMDDhhmmss-hhmm

- Dále jsou možné všechny 3 tvary, kdy za ss je desetinná tečka následována zlomky sekundy.

- Oba typy času jsou odvozeny od typu VisibleString. Jinými slovy: čas se kóduje v ASCII jako bychom jej psali textovým editorem.

- Např.: 19851106210627.3+0545 vyjadřuje 11.6.1985 v 21:06:27,3 místního času v Káthmadú jež je posunut o 5.45 od Greenwich.

  V BER-kódování se UTCTime kóduje v ASCII, tj. uvedený čas bude:
  31 39 38 35 31 31 30 36 32 31 30 36 32 37 2e 33 2b 30 35 35
  (pro orientaci: znak 0 = 30_{16}, znak 1 = 31_{16},..., plus je 2b).

  Musíme ještě doplnit typ (UTCTime má 1716) a délka je 1516, čili:
  17 15 31 39 38 35 31 31 30 36 32 31 30 36 32 37 2e 33 2b 30 35 34 35

Bitový řetězec

- Řetězec bitů se zprava doplňuje výplní tak, aby jeho délka byla násobkem 8 (tj. doplní se na bajty). V DER kódování (na rozdíl od BER) musí být výplň tvořena binárními nulami.
- Zleva se před řetězec doplní bajt binárně vyjadřující, o kolik bitů byl řetězec doplněn. Nejlépe se kódování bitových řetězců pochopí na příkladu:

Příklad :

Řetězec, který se má kódovat 01101110 11

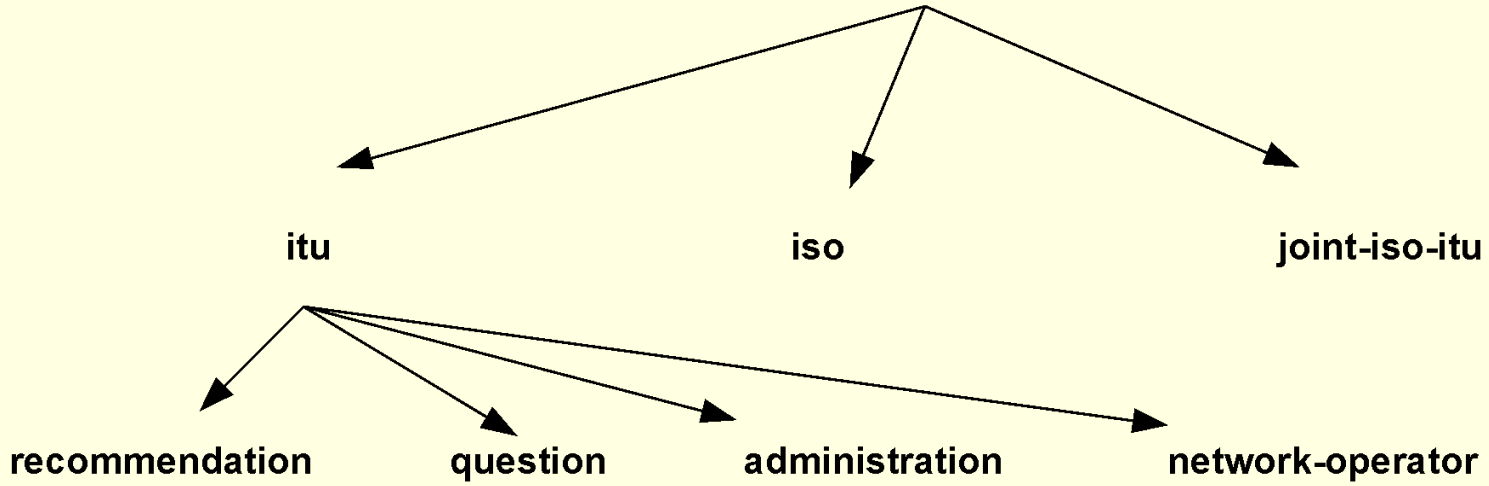
Zprava doplněn šesti binárními nulami na násobek 8:
01101110 11000000

Převeden na bajty (vyjádřeno šestnáctkově) 6E C0

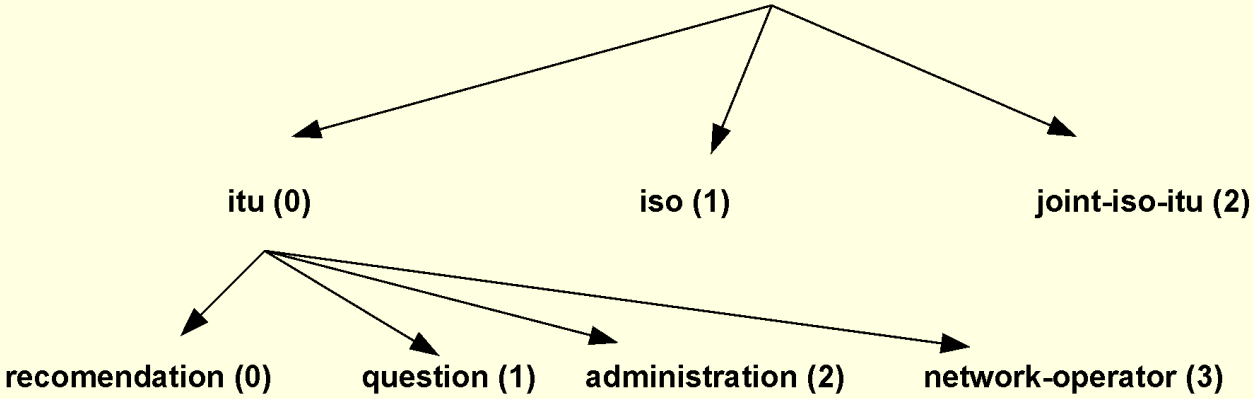
Zleva doplněn bajtem vyjadřujícím délku výplně (06) 06 6E C0

BER kódování (03 je tág pro bit string; 03 je i délka) 03 03 06 6E C0

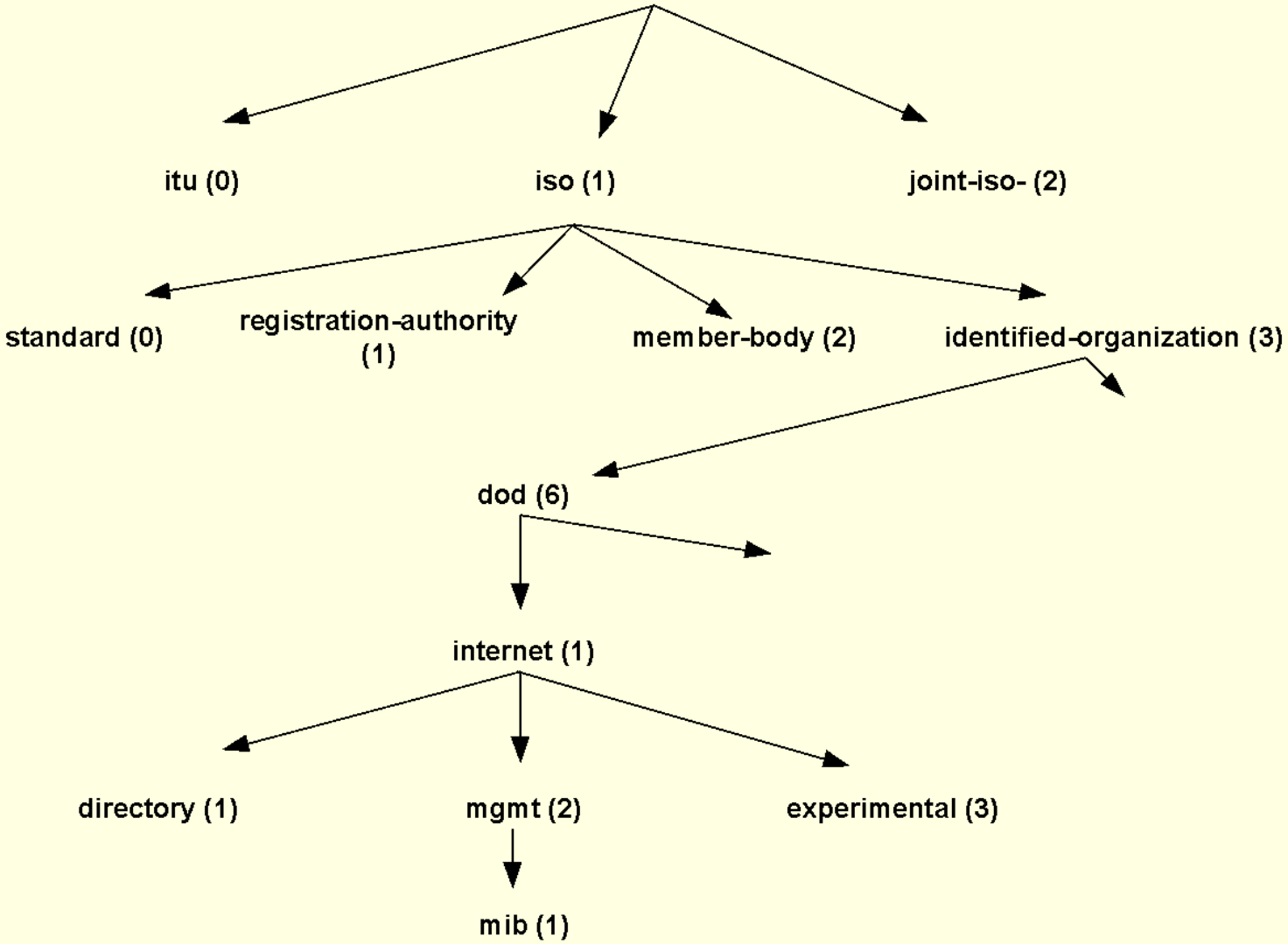
Identifikátory objektů



Identifikátory objektů



Identifikátory objektů



Identifikátory objektů - příklady

- Nové objekty se definují pomocí operátoru ::= . Např. objekt internet se definuje:

- internet OBJECT IDENTIFIER ::= { iso org(3) dod(6) 1}

- nebo jednodušeji:
  internet OBJECT IDENTIFIER ::= { 1.3.6.1}

- Další objekty lze definovat už i za využití podstromu. Např.:
  directory OBJECT IDENTIFIER ::= { internet 1}
  mgmt OBJECT IDENTIFIER ::= { internet 2}
  mib OBJECT IDENTIFIER ::= { mgmt 1}
  atd.

Kódování BER

- Identifikace objektu rsadsi vyjádřená ve složených závorkách je tedy {1.2.840.113549} . Při konverzi do BER se jiným způsobem konvertují první dvě čísla a jiným ostatní:

- Konverze prvních dvou čísel je jiná, protože první číslo nabývá hodnot 0, 1 a 2. Druhé číslo pak hodnot 0 až 39. Provede se výpočet 1. číslo x 40_{10} + 2. číslo.

  Pro náš případ je 1 x 40_{10} + 2 = 42_{10} = $2A_{16}$.

- Následující čísla se pak kódují již samostatně. Jelikož ne každé číslo se vejde do jednoho bajtu, je nutné dlouhá čísla umísťovat do několika bajtů. Jenže jak zjistit, z kolika bajtů se které číslo skládá? Použije se trik. Ke kódování se použije pouze 7 bitů z každého bajtu. Osmý bit (nejvyšší, tj. levý bit) slouží ke specifikaci konce čísla. Pokud je nejvyšší bit nastaven na 1, pak i následující bajt patří k tomuto číslu. Poslední bajt čísla má nejvyšší bit roven nule.

  Problém spočívá v tom, že do každého bajtu se tedy vejde nejvíce 128 hodnot – vše tedy musí být počítáno modulo 128.

Kódujeme 840 jako následující číslo

- 840_{10}= 1101001000_2, takže máme-li k dispozici pouze 7 bitů z každého bajtu, pak z binárních číslic musíme zprava vytvořit skupiny po sedmi číslicích. Dostaneme dvě skupiny 110 a 1001000. První skupiny doplníme vedoucími nulami, pak naše skupiny po sedmi vypadají: 0000110 a 1001000. Nyní musí doplnit vedoucí bit. Po první skupině následuje druhá skupina, takže první skupina obdrží jako vedoucí bit 1. Po druhé skupině již nic nenásleduje, takže druhá skupina obdrží jako vedoucí bit 0. Nyní:
- První skupina bude 10000110 dvojkově, což je šestnáctkově 86.
- Druhá skupina bude 01001000 dvojkově což je šestnáctkově 48.
- Takže desítkové číslo 840 se bude kódovat jako 86 48 (šestnáctkově).

Kódujeme 113549

- 1135949 = 0000110 1110111 0001101 dvojkově (odděleno po sedmicích). První a duhou sedmici doplníme zleva hodnotou 1 a třetí sedmici hodnotou 0. Dostaneme:

- 10000110 = 86 šestnáctkově
- 11110111 = F7 šestnáctkově
- 00001101 = 0D šestnáctkově

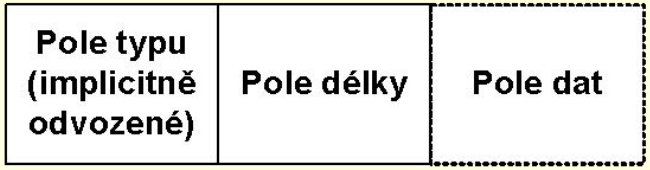
- Výsledkem je tedy 86 F7 0D.

Kódujeme {1.2.840.113549}

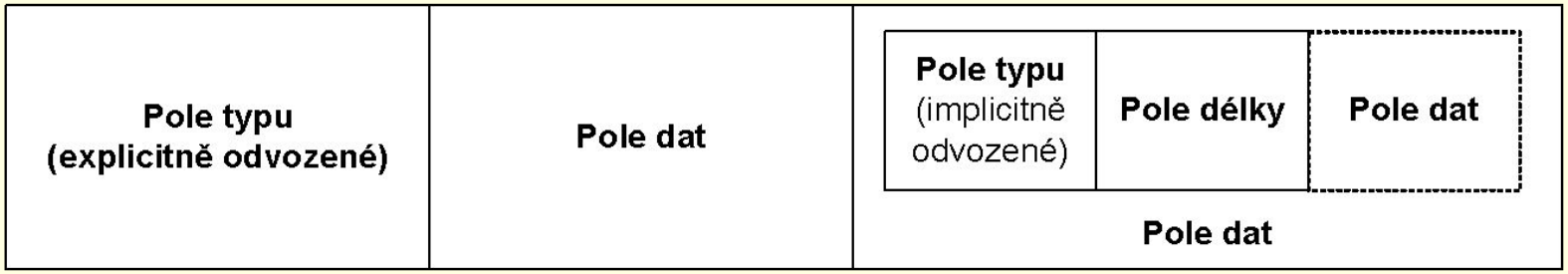
- První dvě čísla {1.2…} se kódují jako 2A (viz před-před-předchozí příklad).
- Další čísle je 840, která se podle před-předchozího příkladu kóduje 86 48.
- Poslední číslicí je 113549, která se podle předchozího příkladu kóduje 86 F7 0D.
- Jako obvykle v BER kódování je výsledkem trojce tág (06 pro identifikátor objektu), délka a hodnota. Hodnota vznikne seřazením jednotlivých kódovaných částí za sebe: 2A 86 48 86 F7 0D. Takže již víme, že délka bude 06.
- Výsledek: **06 06** 2A 86 48 86 F7 0D

Odvozené typy

Implicitně odvozený typ



Explicitně odvozený typ



Příklady implicitně odvozených typů

- ***Příklad 14.14:***

Určete typy pro šířku, výšku a hloubku tak, aby byly platné v rámci vaší firmy (tj. zavádíme privátní typ). Jako tágy si zvolte čísla 1 (výška), 2 (šířka) a 3 (hloubka). Takže se jedná o privátní typy (+ $C0_{16}$), jednoduché, dále nestrukturované typy (+ 0) plus příslušný tág (1, 2 či 3); dostaneme tedy C1, C2 a C3.

- ***Příklad 14.15:***

V kódování BER zapište podle předchozího příkladu předmět vysoký 1 m, široký 2 m a hluboký 1 m.

Výsledkem bude **30 09** c1 01 01 c2 01 02 c3 01 01, tj.:

```
SEQUENCE {
    . [PRIVATE 1]
    . . 01
    . [PRIVATE 2]
    . . 02
    . [PRIVATE 3]
    . . 01
    . }
```

Zápis [PRIVATE 1] znamená, že se jedná o typ třídy PRIVATE s tágem 1.

Příklady – explicitně odvozené typy

- ***Příklad 14.16 (obdoba příkladu 14.14 pro explicitně odvozené typy):***
  Jelikož explicitně odvozený typ je typem konstruovaným, nesmíme zapomenout přičíst šestnáctkově 20.
  Výška bude mít typ C0$_{16}$ (privátní třída) plus 20$_{16}$ (konstruovaný typ) plus 1 (výška), tedy E1. Obdobně šířka E2 a hloubka E3.

- ***Příklad 14.17 (obdoba příkladu 14.15 pro explicitně odvozený typ):***
  V kódování BER zapište podle předchozího příkladu předmět vysoký 1 m, široký 2 m a hluboký 1 m.

  Nyní musíme každý rozměr popsat samostatně. Např. výška se opět bude skládat z trojice tág (E1), délka a pole dat:
  Pole typ je E1 (viz příklad 14.16).

  Pole délka bude obsahovat délku, kterou zjistíme až zkonstruujeme hodnotu.
  Pole data, které bude obsahovat celé číslo (INTEGER) s hodnotou 1, tj. pole data budeme konstruovat z trojice tág, délka a pole dat:

  Tág má hodnotu 2, tj. typ INTEGER,
  Délka je 1
  Pole dat nese hodnotu 1.

  Výsledek je 02 01 01. Výsledek je dlouhý 03 bajty.

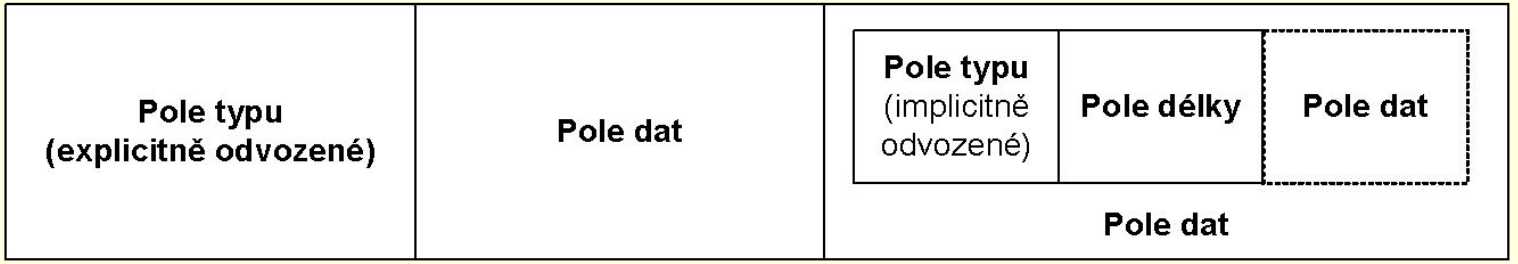
  Čili délka bude **E1 03** 02 01 01. Obdobně šířka bude **E2 03** 02 01 02 a výška **E3 03** 02 01 01.
  Výsledek: **30 0F** E1 03 02 01 01 E2 03 02 01 02 E3 03 02 01 01, tj.:

```
SEQUENCE {
    . [PRIVATE 1] {
    . . INTEGER 1
    . . }
    . [PRIVATE 2] {
    . . INTEGER 2
    . . }
    . [PRIVATE 3] {
    . . INTEGER 1
    . . }
    . }
```

Odvozené typy

Implicitně odvozený typ

Explicitně odvozený typ



Speciální typ CHOICE

- CHOICE umožňuje výběr z několika variant v jazyce ASN.1.
- CHOICE se nokóduje!!!!!!!!!!!!!!!!!!
- Např. v normě PKCS#6 se zavádí struktura (nejedná se o definici certifikátu dle PKI, ale podle PKCS#6):

```
ExtendedCertificateOrCertificate :== CHOICE {
        certificate Certificate,
        extendedCertificate [0] IMPLICIT ExtendedCertificate
      }
```

Tím se chce říci, že struktura ExtendedCertificateOrCertificate je buď

certificate Certificate

nebo

extendedCertificate [0] IMPLICIT ExtendedCertificate

Speciální typ ANY

- Typ ANY označuje jakýkoliv typ.

Příklad opět z certifikátu X.509:

```
AlgorithmIdentifier ::= SEQUENCE {
            algorithm     OBJECT IDENTIFIER,
            parameters  ANY DEFINED BY
  algorithm     OPTIONAL}
```

UTF-8

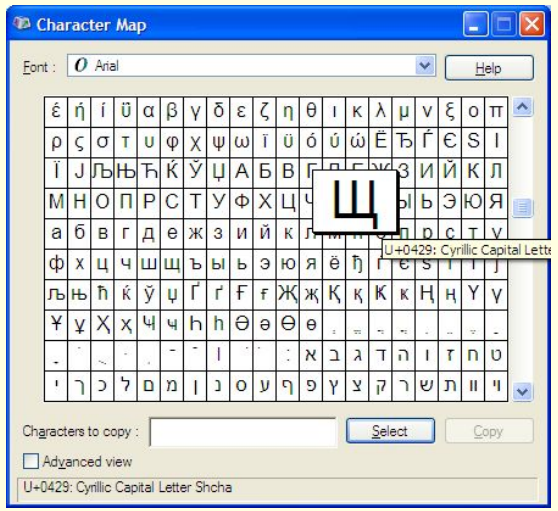
- Unicode
  - UCS-4
  - UCS-2 – též *Basic Multilingual Plane* (BMP).

UTF-8

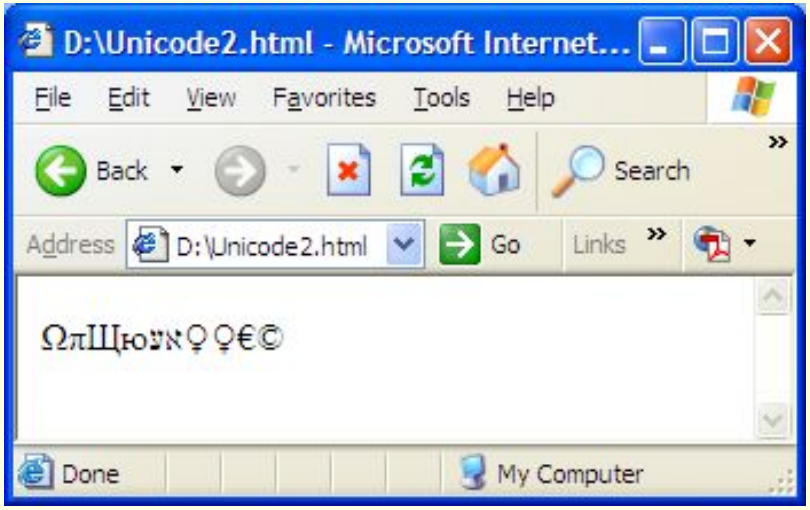
UCS-4 rozsah (šestnáctkově)	UTF-8 binárně
0000 0000-0000 007F	0xxxxxxx
0000 0080-0000 07FF	110xxxxx 10xxxxxx
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx
0001 0000-001F FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx
0020 0000-03FF FFFF	111110xx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx
0400 0000-7FFF FFFF	1111110x 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx 10xxxxxx

Příklad

Název znaku	UCS-2 (Unicode)	UCS-2 binárně	UTF-8 binárně	UTF-8
Velké řecké písmeno ómega	03 A9	1110 101001	**110**01110 **10**101001	CE A9
Malé řecké písmeno pí	03 C0	1111 000000	**110**01111 **10**000000	CF 80
Velké písmeno cyrilice ŠČ	04 29	10000 101001	**110**10000 **10**101001	D0 A9
Malé písmeno cyrilice ju	04 4E	10001 001110	**110**10001 **10**001110	D1 8E
Hebrejské písmeno álef	05 D0	10111 010000	**110**10111 **10**010000	D7 90
Hebrejské písmeno alternativní ajin	FB 20	1111 101100 100000	**1110**1111 **10**101100 **10**100000	EF AC A0
Znak pro ženu	26 40	10 011001 000000	**1110**0010 **10**011001 **10**000000	E2 99 80
Znak Eura	20 AC	10 000010 101100	**1110**0010 **10**000010 **10**101100	E2 82 AC
Znak copyrightu	00 A9	10 101001	**110**00010 **10**101001	C2 A9



Příklad


D:\Unicode2.html - Microsoft Internet...
File Edit View Favorites Tools Help
Back
Search
Address D:\Unicode2.html
Go
Links
ΩπШюאקק€©
Done
My Computer