Formát XB - popis formátu eXtensible Binary Format

Obsah dokumentu

1. Úvod k formátu

Dostává se vám do rukou popis digitálního formátu XB, založeného na principu potenciálně neomezených čísel. Rozhodl jsem se vytvořit tento formát, neboť jsem postrádal formát, který by měl neomezenou dopřednou kompatibilitu a nebyl závislý na použité platformě. Navíc jsem potřeboval ukládat některé méně časté typy dat, jejichž v té době dostupné formáty byly pro mne poněkud nepoužitelné. Rozhodl jsem se tedy vytvořit formát nový a uvolnit jej pod licencí GNU/GPL. Věřím, že výhody, které tento formát přináší převýší náklady na zavedení jeho podpory v programech a operačních systémech. Pokud se chcete podílet na vývoji tohoto formátu, nebo mě upozornit na nějaké chyby, můžete mi napsat email "bomi@zde.cz". Upozorňuji že vesměs nemám s tvorbou tohoto typu zkušenosti, takže je možné očekávat mnoho chyb.

1.1 Cíle

Cílem je vytvořit digitální formát s následujícími vlastnostmi (seřazeno podle priority) :

- k dispozici volně a zdarma pod licencí GNU/GPL
- dopředná kompatibilita
- platformová nezávislost
- jednotná blokově-stromová struktura
- přesvědčivá reprezentace fyzikální reality
- co největší vyjadřovací schopnost
- snadná rozšiřitelnost
- výkonné nástroje volně k dispozici/platformově nezávislé
- předpisy pro výkonné nástroje volně k dipozici
- použité metody bez závislosti na existujících licencích
- podpora vědeckých výpočtů a teoretických modelů
- pevné specifikace pro třídy zařízení

1.2 Licence

Z výše uvedených cílů je snad dostatečně zřejmé, že licence je zcela free, až na části kódů, které jsou k dispozici pod GNU/GPL2. Je zakázáno zahrnout k tomuto formátu jakýkoukoli uzavřenou nádstavbu. Taková nebude schválena a zahrnuta do databáze platných klíčů formátů a nesmí používat logo XB.

1.3 Název formátu

Název formátu je v této fázi určen použitou příponou .XB, a to na eXtensible Binary format. Tento název stejně jako přípona mohou být v budoucnu změněny v závislosti na tom, zda již nějaký formát s tímto názvem existuje, či nikoli. Seznam možných názvů formátu (název potenciální přípony):

- Extensible Binary Format	XB, EB, X2, E2
- Extensible File Format	XF, XO, XI
- Bit Extend Format	BF, BE, BX, 2E, 2X, 2F
- Binary Universal Format	BU, 2U, UF
- Free File Format	3F, FF, FR

Skutečná přípona souboru se nicméně bude skládat z 2 písmen názvu formátu a dalších znaků specifikujících blíže typ obsahu. Mělo by se většinou jednat o první znak v anglickém názvu slova popisujícího typ obsahu. Níže udané přípony jsou pouze orientační a budou pravděpodobně změněny. Seznam přípojných přípon (typ přípona):

- Picture	P
- Bitmap Picture	PB
- Multisize Picture	PM
- Vector Picture	PV
- Animation	A
- Bitmap Animation	AB

- Multisize Animation	AM
- Vector Animation	AV
- Sound	S
- Audio Sound	SA
- Modulable Sound	SM
- Video	V
- Text	T
- Text Document	TD
- Executable	E

1.4 Podpora formátu

Webová stránka pro podporu formátu se nachází na adrese http://bomi.zde.cz/xb, kde je umístěna specifikace. Další informace můžete také získat na oficiální stránce skupiny BOMI: http://bomi.zde.cz V současné době je formát ve stadiu vývoje a existuje dosud pouze jediná aplikace, která bude průběžně zahrnovat jednotlivé typy bloků.

2. Používaný typ čísel

Z důvodů dopředné kompatibility bylo nutné vytvořit takový formát čísla, který by umožňoval ukládat neomezeně velká čísla, případně čísla s potenciálně nekonečnou přesností. Velikost takového čísla by pak byla omezena jen maximální velikostí prostoru, který je počítač schopen alokovat. Použití neomezených čísel je již implementováno v několika programovacích jazycích, ale jejich nevýhodou je větší náročnost výkon počítače. Při vlastním zpracování souboru by měla být data ve vnitřní paměti počítače ukládána přirozeným způsobem. Výsledkem snahy najít takovou formu ukládání čísel je formát BNumber. Tento formát umožňuje ukládat nekonečná přirozená čísla včetně nuly, nehodí se však pro vnitřní reprezentaci ve vnitřní paměti počítače.

2.1. Formát čísel typu BNumber

V době uvedení této verze dokumentu nebylo ještě rozhodnuto konečném tvaru jednotlivých bloků formátu. Před uvolněním první oficiální specifikace bude nutné vybrat nejvhodnější tvar. Také název BNumber (Bit extend NUMBER) může být změněn.

2.1.1. Varianta HighestBitAtByteAsStopBit

Základní tvar čísla je jednobajtový, přičemž hodnota nejvyššího (prvního) bitu je 0. Pokud je nejvyšší bit 1, rozšiřuje se délka čísla o jeden bajt, přičemž se u nového bajtu toto pravidlo aplikuje rekurzivně. Posloupnost hodnot vypadá takto (binární tvar = interval možných hodnot):

```
0xxxxxxx                              = 0..7Fh
1xxxxxxx 0xxxxxxx                     = 0..3FFFh
1xxxxxxx 1xxxxxxx 0xxxxxxx            = 0..1FFFFFh
1xxxxxxx 1xxxxxxx 1xxxxxxx 0xxxxxxx   = 0..0FFFFFFFh
...
```

Jak je vidět, jedná se o relativně jednoduchý model, ve kterém se může člověk relativně snadno orientovat. Je možné se také rozhodnout, který bajt bude určovat jedničku (Big/Little Endian). Nicméně jsou zde i jisté nevýhody:

- přeskočení čísla vyžaduje celé jeho přečtení
- při převodu se musí provádět množství bitových posunů
- délku prostoru nutného pro alokaci lze zjistit pouze přečtením všech bajtů

Jelikož se domnívám, že se jedná o dost nepříjemné vlastnosti, přikláním se osobně ke druhé variantě, respektive k její rozšířené verzi.

2.1.2. Varianta HighestBitsAsByteCount

I tento model má stejný základní jednobajtový tvar, kde nejvyšší bit je 0 a ostatní bity určují číslo. Pokud je nejvyšší bit 1, rozšiřuje se číslo o další bajt, přičemž se počet bajtů rozšiřuje rekurzivně podle dalších nejvyšších bitů. Lépe to lze pochopit z příkladu posloupnosti, která vypadá takto (binární tvar = interval možných hodnot):

```
0xxxxxxx                              = 0..7Fh
10xxxxxx xxxxxxxx                     = 0..3FFF
110xxxxx xxxxxxxx xxxxxxxx            = 0..1FFFFF
1110xxxx xxxxxxxx xxxxxxxx xxxxxxxx   = 0..0FFFFFFFh
...
```

Tato varianta je mnohem přijatelnější, a to jak vzhledem k operaci přeskočení čísla, tak i vzhledem k zjištění místa potřebného k alokaci. Navíc není třeba provádět tolik aritmetických posunů k určení hodnoty čísla. Stačí z čísla odstranit všechny nejvyšší bity až do prvního nulového. Tento typ je nejvíce podobný formátu UTF-8 používaného k ukládání znaků ve standardu Unicode. Je vhodné umístit jedničku na poslední bajt (podobně jako Big Endian), neboť tím odpadává množství aritmetických posunů.

Tím, že se rezervují bity pro záznam délky, vzniká ztráta. Z intervalu, který by bylo možné na daný počet bajtů uložit tak lze umístit méně informace. Tuto ztrátu lze vyjádřit například jako počet nevyužitých bitů na bajt. U obou předchozích variant je tato ztráta konstantní a její hodnota je 1 bit na bajt. Lépe si lze tuto ztrátu ukázat pomocí podílu intervalů na bajt, tedy 1/(2^počet_nevyužitých_bitů_na_bajt). Konkrétně je to v tomto případě 1/2, což znamená, že místo intervalu 0..0FFh je možné v jednom bajtu uložit pouze polovinu, tedy 0..7Fh. Tuto ztrátu lze u velkých čísel snížit použitím následující varianty.

2.1.3. Varianta ExtendedHighestBitsAsByteCount

Třetí a zatím poslední navrhovanou variantou je jednoduché rozšíření předchozí varianty o takzvanou prefixovou formu bajtu. Platí všechny pravidla předchozí varianty, pouze navíc platí, že pokud je hodnota prvního bajtu 0FFh, je před vlastní hodnotu vložen další bajt, který je interpretován jako údaj o délce vlastní hodnoty. Tato hodnota je opět typu ExtendedHighestBitsAsByteCount. Nejlépe je to opět vidět na příkladě.

```
0xxxxxxx                                = 0..7Fh
10xxxxxx xxxxxxxx                       = 0..3FFFh
110xxxxx xxxxxxxx xxxxxxxx              = 0..1FFFFFh
1110xxxx xxxxxxxx xxxxxxxx xxxxxxxx     = 0..0FFFFFFFh
...
11111110 xxxxxxxx .. xxxxxxxx           = 0..0FFFFFFFFFFFFFFh
         \_____ 7 krát _____/
11111111 00000000 xxxxxxxx .. xxxxxxxx  = 0..0FFFFFFFFFFFFFFFFh
                  \_____ 8 krát _____/
11111111 00000001 xxxxxxxx .. xxxxxxxx  = 0..0FFFFFFFFFFFFFFFFFFh
                  \_____ 9 krát _____/
...
```

- obecně pro prefix 0FFh:

```
11111111 N xxxxxxxx .. xxxxxxxx  = 0..2^( 8*(N + 8))
           \____ n+8 krát ____/
```

kde N je další číslo typu BNumber.

Na první pohled se jedná o značné zesložitění, jehož přínosem je pouze zmenšení ztráty u čísel, které se nevejdou do intervalu 0..2^112 a u některých čísel menších se jedná naopak o zvětšení ztráty. Nicméně vzhledem k rekurzi je pak možné mnohem jednodušeji ukládat velmi velká nebo přesná čísla (například pí) a s mnohem menší datovou ztrátou. Průběh ztrát je (počet bajtů vyjadřujících vlastní číslo = množství ztracené informace):

```
1..7          = 0,500000
8             = 0,670123
9             = 0,635129
10            = 0,603149
..
14            = 0,500000
15            = 0.478801
..
127+8         = 0,077761
127+9         = 0,112795
127+10        = 0,112037
...
```

Ztráta se tedy postupně změnšuje (posloupnost konverguje k nule). Nejvyšší ztráta (limes superior) dosahuje hodnoty přibližně 0,67 (na bajt), což je celkem dost velká ztráta. Jiným způsobem, jak je také možné tuto ztrátu snížit je použití základní délky dvoubajtového slova. To by také mohlo vést ke zrychlení a zefektivnění práce s těmito čísly na procesorech s alespoň 16 bitovou datovou sběrnicí (což je dneska již vpodstatě většina), nicméně by to také vedlo k zesložitění celého problému, neboť by se musely mimo jiné, hodnoty ukládat na sudé pozice a provádět další optimalizace. Bajt, jakožto v současné době základní datová jednotka, se zdá být nejvhodnějším kandidátem. Navíc díky frekventovanému používání čísel ze základního intervalu 0..7Fh může tímto způsobem dojít díky používání jednobajtové základní formy k drobné úspoře místa.

2.2. Formy Formátu BNumber

Kromě základního tvaru přirozeného čísla BNatural lze odvodit množství dalších forem. Jedná se především o celočíselný typ se znaménkem a reálný typ. Uveďme si některé z podporovaných tvarů (název - popis):

```
BNatural   - přirozené číslo včetně nuly
BInteger   - celé číslo
BReal      - reálné číslo
BNReal     - nezáporné reálné číslo
BEReal     - reálné číslo s konstantami pro +- nekonečno
BMultiBit  - pole logických hodnot
```

V dalším textu budeme vycházet z varianty ExtendedHighestBitsAsByteCount s jedničkou vpravo. Dále je nutné uvážit dva další podtypy. Jedná se o redundantní a neredundantní podtypy.

2.2.1. Redundantní podtyp

Povolené intervaly jednotlivých binárních tvarů různých délek se prolínají. V této první formě budeme jednoduše tyto intervaly počítat bez dalších úprav, tudíž tímto vzniká jistá nejednoznačnost a redundance. Například číslo 1 můžeme vytvořit v těchto nekonečně mnoha tvarech různých délek:

```
00000001                          = 1
10000000 00000001                 = 1
11000000 00000000 00000001        = 1
...
```

Tato redundance má zřejmě některé výhody, především se s takto zjednodušenými čísly pracuje mnohem jednodušeji, a také lze pomocí této redundance v souboru dopředu rezervovat prostor, pro větší interval hodnot, díky čemuž může dojít o omezení přístupu k souboru. Například při zápisu čísla 200 místo jiného většího (nebo i menšího čísla, pokud má alokované alespoň dva bajty) nemusí nutně dojít ke změně velikosti celého souboru. Na druhou stranu může tato redundance zvětšit velikost souboru bez zvýšení celkové informační hodnoty.
Jednotlivé typy jsou interpretováný takto (typ - popis):

BNatural - Nese prostě celé číslo na určeném počtu bitů
BInteger - Jednoduše je první použitelný bit znaménko. 0 - kladné, 1 - záporné a z důvodu redundace je v podstatě jedno, zda jsou záporná čísla v normálním, inverzním, nebo dvojkovém doplňkovém kódu
BReal - Reálné číslo reprezentují dvě čísla typu BInteger. První číslo je báze a druhé je mantisa, která udává index dvojkového posunu
BNReal - Stejně jako BREAL, akorát je první číslo typu BNatural
BEReal - Reálné číslo s vyznačenými konstantami pro +- nekonečno:

```
00111111  00000000                = +nekonečno
01000000  00000000                = -nekonečno
```

jelikož hodnoty odpovídající těmto konfiguracím lze uvést i v jiné formě, lze tyto konstanty takto stanovit zcela bez problému
BMultiBit - pole bitů indexovaných zprava od 0

Mezi další nevýhody této interpretace patří nejednoznačnost zápisu uvedených typů. Mnohem vhodnější bude zřejmě používání následujícího podtypu.

2.2.2. Neredundantní podtyp

Tento podtyp vznikl odstraněním redundance z redundantního podtypu. Hodnoty jednotlivých typů mají jednoznačné vyjádření, pokud se však hodnota dostane mimo vyhrazený interval musí dojít k přepracování celého souboru, což může vést ke značné režii. Další nevýhodou je pak složitější převod čísel. Jednotlivé typy lze interpretovat takto (typ – popis):

BNatural – Z nejjednoduššího typu BNatural se hodnota získává pomocí algoritmu:

```
Hodnota := X
Pro každé I z intervalu <1..Count> Hodnota:=Hodnota + (2^(I*7))
```

kde X je vlastní hodnota a Count je počet bajtů, které tuto hodnotu vyjadřuje. Uveďme si některé přechody:

```
00000000                              = 0
00000001                              = 1
...
01111111                              = 7Fh
10000000 00000000                     = 80h
10000000 00000001                     = 81h
...
10111111 11111111                     = 407Fh
11000000 00000000 00000000            = 4080h
...
```

Obdobně jsou ošetřeny přechody i vzhledem k prefixu prvního bajtu 0FFh (viz. 2.1.3 Varianta ExtendedHighestBitsAsByteCount)

BInteger – Zahrnutí znaménka je již poněkud složitější operace. Je totiž nutné počítat jinak kladná a jinak záporná čísla. S ohledem na odstranění redundance musí být čísla ukládána ve dvojkovém doplňkovém tvaru. Výpočet pro získání vlastní hodnoty je pak tento:

```
Je-li (Znaménko = "-") potom (
  Hodnota := ( - Neg(X) ) - 1
  Pro každé I celé z intervalu <1..Count> Hodnota:=Hodnota - (2^(I*6))
) jinak (
  Hodnota := X
  Pro každé I celé z intervalu <1..Count> Hodnota:=Hodnota + (2^(I*6))
)
```

kde X je vlastní hodnota bez znaménka a Count je počet bajtů, které tuto hodnotu vyjadřuje. Operace Neg je inverze platných bitů v X. Uveďme si některé přechody:

```
...
11011111 11111111 11111111            = -2041h
10100000 00000000                     = -2040h
...
10111111 11111110                     = -42h
10111111 11111111                     = -41h
01000000                              = -40h
...
01111110                              = -2
01111111                              = -1
00000000                              = 0
00000001                              = 1
...
00111111                              = 3Fh
10000000 00000000                     = 40h
10000000 00000001                     = 41h
...
10011111 11111111                     = 203Fh
11000000 00000000 00000000            = 2040h
...
```

BReal - Nejběžnější způsob reprezentace reálných čísel je pomocí dvou celých čísel, přičemž jedno z nich určuje bázi a druhé mantisu. Například procesorů s architekturou Intel jsou reálná čísla realizována pomocí standardu IEEE 764, který používá k odstranění redundance metodu "neviditelné" jedničky. Tuto metodu lze s výhodou použít i v tomto případě. Je možné volit mezi umístěním jedničky před nejvyšší, nebo za nejnižší bit. Mimo jiné je zde také možnost několika interpretací mantisy. Obě čísla lze chápat ve smyslu předchozího typu BInteger. Popišme tedy nejprve techniku umístění "neviditelné" jedničky před nejvyšší bit báze. Při interpretaci čísla je pak nutné po případné negaci záporného čísla přidat před platné bity jeden bit hodnoty 1. Už v této fázi je vidět jisté obtíže s přeskakováním znaménka. Dalším problémem je pak hned interpretace posunu "polovinné" (dvojkové) tečky. Ukažme si tři ze způsobů interprece:

```
 - interpretace tečky k nejspodnějšímu bitu
  00000000  00000000                  = 64              [(1)000000.]
 - interpretace tečky k nejvyššímu bity
  00000000  00000000                  = 1               [(1).000000]
 - interpretace tečky před "neviditelnou" jedničku
  00000000  00000000                  = 0.5             [.(1)000000]
```

Jak je vidět, výhodnější je umístit tečku k nejvyššímu bitu, s čímž ale souvisí problémy s přepočtem mantisy v závislosti na délce vlastní hodnoty báze, kterým bychom se však nevyhnuli ani použitím umístěním tečky na konec slova, kdy by zase bylo nutné nějakým způsobem posouvat mantisu. Třetí způsob je vpodstatě zcela nevhodný. Vhodnější asi bude používat méně neobvyklou techniku přidání jedničky na konec čísla. Opět si uvedeme způsoby interpretace tečky:

```
 - interpretace tečky k nejvyššímu bity
  00000000  00000000                  = 0.015625        [.000000(1)]
 - interpretace tečky k nespodnějšímu bitu
  00000000  00000000                  = 0.5             [000000.(1)]
 - interpretace tečky za "neviditelnou" jedničku
  00000000  00000000                  = 1               [000000(1).]
```

Tento na první pohled kostrbatý způsob umístění jedničky má několik výhod a nevýhod. Zřejmě nejvhodnější způsob je třetí z uvedených. Je nutné stanovit speciální konstantu pro nulu. Nabízí se využít konstantu odpovídající nule ve formátu BInteger i BNatural. Takováto volba vede k tomuto algoritmu:

```
 Je-li (X=0 a zároveň Y=0) potom Hodnota:=0 jinak (
   Hodnota:=(X*2+1)*(2^Y)
   Je-li (X>0 a zároveň Y=0) potom Hodnota:=Hodnota - 2
 )
```

Opět si uveďme některé konstanty a přechody. Pokud je za hodnotou uvedena druhá hodnota v závorce, pak tato hodnota odpovídá rozdílu oproti interpretaci bez hodnoty 0.

 - čísla s mantisou nula:

```
  ...
  10111111 11111111  00000000         = -81h
  01000000  00000000                  = -7Fh
  01000001  00000000                  = -7Dh
  ...
  01111110  00000000                  = -3
  01111111  00000000                  = -1
  00000000  00000000                  = 0 (1)
  00000001  00000000                  = 1 (3)
  00000010  00000000                  = 3 (5)
  ...
  00111111  00000000                  = 7Dh (7Fh)
  10000000 00000000  00000000         = 7Fh (81h)
  ...
```

 - další čísla s různými mantisami:

```
  01111111  00000001                  = -2
  00000000  00000001                  = 2
```

```
00000001  00000001                    = 6
00000010  00000001                    = 10
00000000  00000010                    = 4
00000000  00000011                    = 8
00000000  01111111                    = 0.5
00000001  01111111                    = 1.5
```

Zřejmě převod čísla z nativního tvaru do BReal je poněkud obtížnější. Je totiž nutné dělit číslo dvěmi, dokud nebude zbytek 1, nebo násobit, dokud nebude necelá část čísla rovna 1/2. Takto vzniklé celé číslo se pak uloží do báze a počet dělení, nebo - počet násobení se poté uloží do mantisy. Tento postup bude vhodné v budoucnu co nejvíce optimalizovat. K výhodám zvolené formy patří i to, že parita mantisy určuje, zda se jedná o celé číslo, nebo číslo, které má i necelou část.

BNReal - Realizuje se stejně jako BREAL, pouze první číslo je typu BBYTE

BEReal - Reálné číslo s vyznačenými konstantami pro +- nekonečno:

```
00111111 00000000                    = +nekonečno
01000000 00000000                    = -nekonečno
```

Zřejmě lze použít stejné konstanty vybrané pro redundantní verzi. Tentokrát však to již není proto, že by odpovídající hodnoty bylo možno vyjádřit jinak. Je nutné upravit převod čísla a to takto:

```
Je-li (X=0 a zároveň Y=0) potom Hodnota:=0 jinak
Je-li (X=3Fh a zároveň Y=0) potom Hodnota:=+nekonečno jinak
Je-li (X=-40h a zároveň Y=0) potom Hodnota:=-nekonečno jinak (
  Hodnota:=(X*2+1)*(2^Y)
  Je-li (X>0 a zároveň Y=0) potom Hodnota:=Hodnota - 2 jinak
  Je-li (X>3Fh a zároveň Y=0) potom Hodnota:=Hodnota - 2 jinak
  Je-li (X<-40h a zároveň Y=0) potom Hodnota:=Hodnota + 2
)
```

Tímto relativně jednoduchým postupem se uvolní dané konstanty posunutí významu ostatních.

BMultibit - pole bitů indexovaných zprava od 0. Stejné jako u redundantní formy

V dalším textu budeme vždy používat tyto typy právě v tomto uvedeném podtypu.

3. Struktura formátu

Každý soubor začíná třemi konstantními znaky 'XB1'. Za těmito znaky následují jednotlivé bloky. První blok se nemusí vyskytovat, pak se soubor nazývá prázdný. Pokud se v souboru první blok vyskytuje, určuje jeho typ typ celého dokumentu. Každý blok se skládá ze tří částí. První část se nazývá hlavička a obsahuje tři hodnoty:

BNatural – BlockType
BNatural – DataSize
BNatural – SubBlocksSize

Hodnota BlockType specifikuje typ bloku podle databáze platných klíčů (viz. Typ Bloku), DataSize určuje délku druhé tzv. datové části v bajtech. SubBlockSize určuje délku části obsahující podřízené objekty. Pokud existuje rozpor mezi SubBlockSize a vlastními délkami bloků, je celý soubor neplatný. Za všemi bloky se nachází tzv. AdvancedData, která nemá omezenou délku.

3.1. Typ bloku

Typ bloku je přirozené číslo, které uvádí typ bloku (dle specifikace formátu). Prvních 407Fh kombinací je vyhraženo pro vnitřní potřebu formátu, zbývající indexy jsou k dispozici pro volné použití. Typ bloku určuje strukturu datové části. Ta se skládá z jednotlivých položkek, které jsou obecně typu BNatural. Jedinou vyjímkou je blok typu Obecný datový blok (viz. níže). U všech ostatních bloků platí, že má tyto první dvě hodnoty:

BNatural – Major Version
BNatural – Minor Version

Tyto konstanty určují úplný význam druhé části bloku, zatímco BlockType určuje pouze základní význam. Navíc se podle těchto konstant interpretují všechny další hodnoty v datovém bloku, povolené maxima, stejně jako povolené typy bloků v bloku podřízených bloků.

Důvodem k uvádění dvou čísel verzí je snaha o implementace zpětné kompatibility. Programům používajícím formát XB ukládá specifikace jistá striktní pravidla. Pokud se čtený soubor liší od podporované verze pouze v hodnotě Minor Version, může aplikace soubor otevřít, a to přeskočením nepodporovaných hodnot a bloků. Pokud při tomto procesu narazí na překročení meze, je vstupní soubor neplatný. Pokud se soubor liší v hodnotě Major Version nesmí program soubor interpretovat. Musí být použita nová verze programu, nebo staženy nástroje pro konverzi.

3.2. Specifikace verzí

Jednotlivé verze bloků se nemusí nutně lišit v počtu a tvaru proměnných. Z důvodu lepší hardwarové podpory umožňuje formát specifikovat takzvanou omezenou normu, která omezuje počty bloků a hodnoty tak, aby dané soubory mohly být používány zařízeními s konstantní kapacitou paměti.

4. Základní typy bloků

Bloky tohoto typu jsou by měli být používány jako základní bloky pro ukládání dat ve všech dalších blocích.

4.1. Obecný datový blok

Obecný datový blok je základní typ bloku, který umožňuje ukládání posloupnosti dat. Velikost části podřízených bloků je striktně 0 a velikost datové části může být libovolné přirozené číslo včetně 0. Pokud je velikost datové části 0, hovoříme o takzvaném prázdném bloku. Pokud je nenulová, jsou jednotlivé bajty interpretovány pole bajtových prvků. Obecný datový blok je tedy předpis přiřazující celému číslu z intervalu <0..DataSize-1> číslo z intervalu <0..255> a je rozumné hovořit o něm jako o poli, nebo také jako o jednořádkové matici, případně jako o funkčním symbolu arity 1. Konstanta BlockType má hodnotu 0. Platí tedy, že tento blok je předpis:

<0..DataSize-1> -> <0..255>

V době vydání této specifikace je tento blok jediný, který má pevně stanovenou konstantu BlockType. U všech dalších bloků je vyhrazena možnost změnit index před první veřejnou verzí.

4.2. Odkaz

Jednotlivé bloky jsou číslovány dle pořadí, ve kterém jsou uvedeny. Tento typ pak umožňuje odkazovat se na některý z těchto bloků ve stejném nebo jiném souboru ať již v místní složce, nebo kdekoli jinde v sítí, nebo internetu. Jedná se tedy o blok, který při použití vrací odkazovaný blok. V základní verzi je způsob získání tohoto bloku plně přenechán operačnímu systému. A tedy to, zda bude blok stáhnut z internetu, nebo bude použita jeho verze uložená v místní vyrovnávací paměti by nemělo být postatné.

- Verze 0: File Inner Path
  Adresace položky v rámci souboru. Hodnoty jsou následující:

  BNatural – UpCount
  BNatural – FileIndex0
  BNatural – FileIndex1
  BNatural – FileIndex2
  ...

  Tyto hodnoty jednoznačným způsobem určují cestu k bloku. Hodnota UpCount = 0 odpovídá odkazu na nadřazený blok. Pokud hodnota UpCount přesáhne hloubku kořenové hloubky souboru, nebo pokud kterákoli z hodnot FileIndex přesáhne počet bloků na dané úrovni, je celý odkaz považován za neplatný. Nutno ještě zmínit případ, kdy odkaz odkazuje na jiný odkaz, což není obecně zakázáno. V takovém případě je nutné ověřovat posloupnost odkazů na zacyklení.
- Verze 1: Basic FileSystem Path
  Jedná se o adresaci položky, nebo souboru v rámci souborového systému. Omezení je kladeno pouze na zákaz přesáhnutí takzvané kořenové složky. Tato složka je odlišná v operačních systémech Windows a Unix. Zatímco u operačních systémů Windows se jedná o kořen jednotky "X:\", v systémech Unix se jedná o kořenovou složku "/". Cesta je opět relativní:

  BNatural – UpCount
  BNatural – TextPathIndex
  BNatural – FileIndex0
  BNatural – FileIndex1
  BNatural – FileIndex2
  ...

  Hodnota UpCount je tentokrát omezena na zákaz přesáhnutí hloubky kořenové složky, přičemž mezi složkami operačního systému a částí závislých bloků je plynulý přechod ve formě tzv. kořenového bloku, jehož nadřazenou složkou je složka operačního systému. Kromě toho existují další neplatné formy zápisu. Jsou to především kolizní situace, kdy je použit FileIndex a TextPathIndex neukazuje na soubor kompatibilní s formátem XB. Další podmínkou je, že pokud Hodnota UpCount "nedosáhne" mimo vlastní soubor, musí být blok odpovídající TextPathIndex prázdný. Tím se dostáváme k významu hodnoty TextPathIndex, která určuje index bloku v části podřízených bloků, který je typu (nebo vrací typ) posloupnosti textových řetězců. Platí také, že tento odkaz může ukazovat na soubor, nebo složku v souborovém systému. Kromě toho je zřejmé, že verze 0 je podmnožinou verze 1.

- Verze 2: Protokol Path
  Další z podverzí je odkaz na soubor nebo blok adresovaným určitým protokolem. Jedná se opět o rozšíření předchozí verze, tentokrát s použitím bloku Protokol. Seznam proměnných vše osvětlí:

```
BNatural – FileProtocolIndex
BNatural – UpCount
BNatural – TextPathIndex
BNatural – FileIndex0
BNatural – FileIndex1
BNatural – FileIndex2
...
```

  Zpětnou "kompatibilitu" s verzí 1 nyní zajišťuje tzv. prázdný protokol, neboli protokol "internal" (viz. Komunikační protokol). Nově je tedy možnost realizovat kořen cesty pomocí takzvaného protokolu, což je způsob, jak umožnit přístup k libovolnému souboru použitím stávacích technologií, jako jsou například protokoly http, ftp, smb a další.
- Verze 3: Link Elongation
  Další z verzí je už pouze kosmetickou úpravou. Jedná se o takzvané navázání na vnitřní link:

```
BNatural – LinkIndex
BNatural – UpCount
BNatural – TextPathIndex
BNatural – FileIndex0
BNatural – FileIndex1
BNatural – FileIndex2
...
```

4.3. Textový řetězec

Při ukládání textu je nutné vzít do úvahy podporu libovolného jazyka, kódování a dalších vlastností textu. Text je totiž jistá forma komprese.

4.3.1. Jazyk

Tento blok slouží pro určení jazyka, ve kterém je uložen text, případně obrázek, nebo jiná jazyková data. Pro základní verzi byla použita dostupná definice čísel pro jednotlivé světové jazyky. Další verze bloku jsou rezervovány pro budoucí použití (například mimozemské jazyky).

- Verze 0: RFC LanguageNumber
  Tento blok využívá pro specifikaci jazyka normalizovaných čísel uvedených na internetu, například na stránkách IANA.

```
BNatural – Major Language Number
BNatural – Minor Language Number
```

  Pozn. Pro verzi 0.1 připadá v úvahu přidání indexu pro místní pozici ve vesmíru / jazykovou skupinu.
- Verze 1: RFC LanguageString
  Tato verze využívá specifikace názvů jazyků.

```
BNatural – StringIndex
```

4.3.2. Kódování znaků

Určení kódovaní textu pomocí interpretativní tabulky významu do univerzálního jazyka (dosud neexistuje?).

- Verze 0: IANA CharacterSet Number
  Využití indexů znakových sad z internetu.

  BNatural - Major CharacterSet Number
  BNatural - Minor CharacterSet Number

- Verze 1: IANA CharacterSet String
  Tato verze využívá specifikace kódování podle jednoznačné identifikace znakovým řetězcem.

  BNatural - StringIndex

5. Základní proměnné

Tato sada bloků slouží pro ukládání některých matematických hodnot.

5.1. Číslo

Tento blok se používá pro ukládání základních čísel především ve spojitosti s konfiguracemi a matematickými modely.

- Verze 0: Basic Number
  Používá první proměnnou pro určení typu hodnoty a dále následuje určitý počet dalších proměnných obsahujících vlastní hodnotu.

  BNatural – NumberType
  ? – Value

  Proměnná NumberType může v budoucnu určovat některou z hodnot: BNatural, BInteger, BReal, BNReal, BNNatural, BNInteger, BMultiBit, BComplex, BCustom

5.2. Časový údaj

Pro ukládání časového údaje je možné využít několik forem ukládání data a času, jako jsou například GMT, nebo Timestamp.

- Verze 0: Basic Time
  Proměnná TimeType by měla určovat například některý z typů: Timestamp, GMT, DOSTime

  BNatural – TimeType
  BNatural – TimeValue

5.2. Poloha

Tento blok umožňuje určit polohu v prostoru. Využívá k tomu index planet (asi neexistuje) a adresu skládající se z indexů oblastí, případně planetární souřadnici.

6. Bitmapový obrázek

Jako jedno z prvních použití formátu bylo vybráno ukládání bitmapových dat, neboť se jedná hojně ukládaná data, jejichž vyjádření v textové formě není příliš vhodné.

6.1. Blok bitmapového obrázku

Tento blok určuje, že data ukládaná v podblocích reprezentují obrázek a uvádí povolené typy podbloků. Povolené bloky jsou zatím: vytvoření, úprava, přístup, rovina. Uveďme alespoň specifikaci bloku rovina.

6.2. Rovina odrazu a pohlcení

Tato rovina je použitelná především pro opticky zobrazitelná data. Jedná se o trinární funkci, která vrací dvě reálné hodnoty v intervalu <0,1>, které určují indexy odrazu a pohlcení. Tři vstupní parametry určují souřadnice a frekvenci.

- Verze 0: Standard AbsorbingPlane
  Tato základní forma používá pouze několik základních vlastností:

  BNatural – RatioIndex
  BNatural – FrequencyMapIndex

6.2.1 Bodový poměr

Bodový poměr určuje poměr jedné vzdálenostní jednotky v rovině vzhledem ke skutečné velikosti ať už na obrazovce, tiskárně nebo jiném výstupním zařízení. V současné době se používá stále ještě větší množství jednotek, ať už jsou to centimetry, palce nebo jejich různá odvození.

6.2.2 Frekvenční mapa

Tento blok určuje způsob interpretace jedné jednotky a výšku základní frekvence pomocí fyzikálních jednotek, především pomocí základních.

Poznámky pro další vývoj:

Dalším omezením maximální velikosti čísel může být specifikační omezení. (viz. Specifikace verzí)

Verze bloků - specifikace
Rezervace indexů
Digitální efekty
Detekce souboru
Kompatibilita
Vztažné jednotky

Možné hodnoty BlockType
0 - obecný datový typ
1 - zástupce cesty k objektu
2 - množina objektů
3 -

Další bloky :
CRC blok
Crypt blok
Compressed blok
Bitmapový obrázek
Vektorový obrázek
Bitmapový prostor
Vektorový prostor
Animace (Čas. konst. + obr.)
Video (Čas. konst. + obr. + zvuk)
Font (obrázek + index. tab.)
Textový soubor
Zvuková stopa
Časová konstanta
Délková konstanta
Indexová tabulka
Databázová položka
Poloha bodu
Energetický paprsek (... rádio ... světlo ... gama ...)
Spustitelný program

Apendixy:
IANA Character Sets
Language Numbers