



XBFF - popis eXtensible Binary File Format

Obsah dokumentu

1. [Úvod k formátu](#)
 - 1.1 [Cíle](#)
 - 1.2 [Licence](#)
 - 1.3 [Název formátu](#)
 - 1.4 [Podpora formátu](#)
 - 1.5 [Odkazy](#)

2. [Neomezená čísla](#)
 - 2.1. [Formát BNumber](#)
 - 2.1.1. [Varianta HighestBitAtByteAsStopBit](#)
 - 2.1.2. [Varianta HighestBitsAsByteCount](#)
 - 2.1.3. [Varianta ExtendedHighestBitsAsByteCount](#)
 - 2.2. [Formy Formátu BNumber](#)
 - 2.2.1. [Redundantní podtyp](#)
 - 2.2.2. [Neredundantní podtyp](#)

3. [Struktura formátu](#)
 - 3.1. [Hlavička souboru](#)
 - 3.2. [Blok](#)
 - 3.2.1. [Hlavička bloku](#)
 - 3.2.2. [Datová část bloku](#)
 - 3.2.3. [Část podřízených bloků](#)
 - 3.3. [Zpracování souboru](#)
 - 3.3.1. [Platnost souboru](#)
 - 3.3.2. [Pravidla platnosti](#)
 - 3.3.3. [Pravidla kompatibility](#)
 - 3.3.4. [Specifikace verzí](#)
 - 3.3.5. [Vyhodnocení typu bloku](#)
 - 3.3.6. [Doporučený postup zpracování](#)
 - 3.4. [Přístupové metody](#)
 - 3.4.1. [Metoda GET](#)
 - 3.4.2. [Metoda READ](#)
 - 3.4.3. [Metoda HANDLE](#)

4. [Systémové bloky](#)
 - 4.1. [Obecný datový blok](#)
 - 4.2. [Odkaz](#)
 - 4.3. [Specifikační blok](#)
 - 4.4. [Protokol](#)

5. [Základní řídicí bloky](#)
 - 5.1. [Určující typ](#)

5.2. Určující skupina

5.3. Odkaz

6. [Bitmapový obrázek](#)

6.1. Blok bitmapového obrázku

6.2. Rovina odrazu a pohlcení

6.2.1 Bodový poměr

6.2.2 Frekvenční mapa

7. [Zvukový soubor](#)

8. [Podpůrné aplikace a knihovny](#)

8.1. [Knihovna BNumLib](#)

8.2. [Knihovna XBLib](#)

8.3. [Editor formátu XB](#)

[Co je nového?](#)

[Připravuje se...](#)

Copyright: © BOMI, Miroslav Hajda <http://bomi.zde.cz> bomi@zde.cz

Version: 1.0 work release 4

Last updated: 2003.09.14

Language: Czech

1. Úvod k formátu

Dostává se vám do rukou popis experimentálního datového digitálního formátu XB. Základem tohoto formátu je systém pro ukládání potenciálně neomezených čísel. Rozhodl jsem se vytvořit tento formát, neboť jsem nebyl spokojen se současnými binárními formáty a nenašel jsem žádný natolik komplexní, který by mě vyhovoval. Věřím, že mnou vytvořený formát bude přínosný pro rozvoj v oblasti informačních technologií a že výhody které přináší, převýší náklady na zavedení jeho podpory v programech a operačních systémech. Ačkoli je možné že tento projekt nebude úspěšně dokončen, věřím, že by se mohl někomu jinému stát inspirací pro vytvoření binárního formátu nové generace, schopného implementovat všechny pokročilé techniky.

Pokud byste se chtěli podílet na vývoji tohoto formátu, nebo mě upozornit na nějaké chyby, můžete napsat zprávu na email bomi@zde.cz. Upozorňuji, že s tvorbou dokumentů tohoto typu nemám zkušenosti, takže se jistě dopustím mnoha chyb. Věřím, že tyto chyby budou v první uvolněné verzi již odstraněny.

1.1. Cíle

Jistě jste se již setkali značkovacími jazyky, jejichž obliba v poslední době rychle roste. Tyto značkovací jazyky, jako je například [XML](#) mají řadu nesporných výhod. Za všechny jmenujme například výbornou rozšiřitelnost, nebo například výbornou čitelnost. Rozmach dosáhl již takové úrovně, že jsou tyto jazyky používány i pro reprezentaci některých tradičně binárních dat, jako jsou například obrázky, nebo zvukové soubory. S tímto přístupem nesouhlasím, neboť popírá skutečný význam dat a znemožňuje účinnou kompresi. Při představě obrázku uloženého jako záznam znaků 0..9, A..F se mi prostě dělá špatně... Angličtinu, která je v této oblasti nejpoužívanější a pro tyto účely vhodná díky své nepřilíhš rozsáhlé abecedě a jednoduché gramatice, rozhodně nemůžeme považovat za natolik nadčasový jazyk, že by u něj nedocházelo k vývoji. Přestože díky konzervativnosti rodilých anglický mluvčích nedochází k příliš častým změnám, lze přesto očekávat, že i tento jazyk bude překonán. Mnohem vhodnější se mi jeví používání řeči čísel, která dle mého názoru nikdy neztratí svou logiku, a které jistě budou rozumět i mimozemské civilizace s nimiž se snad v budoucnu setkáme. :-)

Uved'me si některé z cílů:

- Volně a zdarma dispozici, případné programové části pod licencí [GNU/GPL2](#)
Věřím, že již minula doba, kdy se komerční společnosti domnívali, že digitální formáty mohou být stejně jako části programového kódu v soukromém vlastnictví. Rychlý vývoj v oblasti informačních technologií ukázal velmi krátkou životaschopnost takto uzavřených a nedokumentovaných formátů a celkovou nevýhodnost tohoto přístupu. Přestože stále existují výjimky (např. [Microsoft Corporation](#)), je dle mne dostupnost dokumentace nutností.
- Kompatibilita a snadná rozšiřitelnost
Ani tyto samozřejmnosti dnes stále ještě nepatří mezi všeobecně podporované vlastnosti, a proto považuju za vhodné se o nich zmínit. Pod pojmem kompatibilita je nutné chápat jak dopřednou, tak zpětnou přenositelnost. Rozšiřitelnost je naproti tomu vlastnost, kterou lze s výhodou přejmout ze značkovacích jazyků (např. [DTD](#))

- **Blokově-stromová struktura**
Nejvhodnější matematická struktura pro digitální formát by zřejmě mělo být členění do bloků umístěných ve stromu. Tento přístup se již osvědčil u mnoha současných formátech a zdá se být nejvhodnější.
- **Jednotná struktura**
Věřím, že pokud se povede navrhnout dostatečně kvalitní systém pravidel, bude mít vytvořený formát dlouhodobě stálý tvar a nebude nutné provádět časté úpravy, které vedou k nutnosti vynakládat další náklady.
- **Platformová nezávislost a reprezentace fyzikálních skutečností**
Digitální formát by neměl reprezentovat data tak, aby byla výhodná pro zpracování danou aplikací, ale zobrazení by mělo odrážet to, co dané data ve skutečnosti představují.
- **Podpora vědeckých výpočtů a teoretických modelů**
Binární matematický formát by měl umět ukládat různé matematické modely a jiné matematické struktury. Měl by také reflektovat nejnovější vědecké poznatky a umožnit jejich co nejrychlejší využití v praxi.
- **Dostupnost základních podpůrných aplikací**
Většina těchto cílů je samozřejmě nemyslitelná bez zázemí aplikací, které by daný formát používali a kvalitních nástrojů pro převod mezi stávajícími formáty. Věřím, že se podaří přesvědčit konzorcia firem o výhodnosti kvalitního jednotného formátu (pokud takový bude) a tyto firmy jej implementují do svých produktů.
- **Snadná aktualizovatelnost K dispozici by mělo být internetové centrum, které by mělo umožnit aplikacím používající formát snadno získat nové verze knihoven. K dispozici by mělo být několik verzí - dll, Java class, so a další.**

1.2. Licence

Tento formát bude k dispozici zcela zdarma až na části nutných obecných programových kódů, které budou šířeny pod licenci [GNU/GPL2](#). Tento volný způsob šíření by se neměl nikdy změnit, neboť by se jednalo o porušení základních cílů formátu, tudíž je formát zdarma i pro komerční použití. Je však zakázáno rozšiřovat formát o vlastní struktury, které by byly v rozporu s danými pravidly. Takové rozšíření nebude schváleno a nesmí používat oficiální značku formátu. Do budoucna se počítá s možnou změnou licence v důsledku nevhodnosti, případně nutnosti úprav - vše však v duchu volné dostupnosti s omezením rozšiřitelnosti s cílem udržení čistoty.

1.3. Název formátu

V současné vývojové fázi nebyl dosud název formátu pevně vybrán. Je to z toho důvodu, aby nedošlo ke kolizi s názvem nebo příponou nějakého již existujícího formátu a také se počítá s možností vývoje komerčně úspěšného loga. Prozatím byl zvolen pracovní název eXtensible Binary Format s ohledem na v současné době ve výpočetní technice nejpoužívanější jazyk Angličtinu a popularitu písmene X. :-) (také byl vzat ohled na logiku daného názvu) Tomuto názvu odpovídá přípona ".xb".

Seznam možných názvů formátu (název formátu, 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)

Přípona souboru se bude skládat ze 2 písmen předpony a dalších znaků, které budou blíže určovat typ obsahu souboru. Mělo by se jednat o první znak anglického slova popisujícího daný typ. Orientačně uvádím seznam několika možných přípon, které však nejsou konečné.

Seznam možných 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**
 - **Sound Stream SS**
- **Video V**
 - **Video Stream VS**
- **Text T**
 - **Text Document TD**
- **Executable E**

Následně bude nutné zvážit textové označení formátu MIME a případnou nutnost odlišování začátku souboru podle typu obsahu, tedy "něco/x-xb".

1.4. Podpora formátu

V současném vývojovém stadiu neexistuje žádná aplikace, která dokáže formát zpracovávat. Průběžně bude vyvíjena knihovna pro práci s formátem (pravděpodobný název XBLib, nebo XBFFLib) a program pro základní konverze a úpravy. Informace o těchto knihovnách a dalších podpůrných aplikacích lze získat v části [Podpůrné aplikace a knihovny](#).

1.5. Odkazy

[\[http://bomi.zde.cz/xb\]](http://bomi.zde.cz/xb) - Oficiální webová stránka projektu

[\[http://bomi.zde.cz\]](http://bomi.zde.cz) - Stránka vývojové skupiny BOMI group.

2. Neomezená čísla

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 a jejich hlavní 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 uložit libovolné přirozené číslo včetně nuly, příliš se však nehodí pro reprezentaci ve vnitřní paměti počítače.

2.1. Formát BNumber

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

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..1FFFFFFh
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.

Příklad (binární tvar = interval možných hodnot):

```

0xxxxxxx          = 0..7Fh
10xxxxxx xxxxxxxx = 0..3FFF
110xxxxx xxxxxxxx xxxxxxxx = 0..1FFFFFF
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 ([Big Endian](#)), neboť tím odpadáva 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 $1/(2^{\text{počet_nevyužitých_bitů_na_bajt}})$. 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 řekněme "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.

Příklad:

```

0xxxxxxx          = 0..7Fh
10xxxxxx xxxxxxxx = 0..3FFFh
110xxxxx xxxxxxxx xxxxxxxx = 0..1FFFFFFh
1110xxxx xxxxxxxx xxxxxxxx xxxxxxxx = 0..0FFFFFFFh
...
11111110 xxxxxxxx .. xxxxxxxx          = 0..0FFFFFFFFFFFFFFFFh
    \_____ 7 krát _____/
11111111 00000000 xxxxxxxx .. xxxxxxxx = 0..0FFFFFFFFFFFFFFFFh
    \_____ 8 krát _____/
11111111 00000001 xxxxxxxx .. xxxxxxxx = 0..0FFFFFFFFFFFFFFFFh
    \_____ 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 zesložnění, 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 π) a s mnohem menší datovou ztrátou. Ne vždy se také musí používat slova 8-bitové délky a čím kratší slovo a větší hodnota, tím je tato metoda výhodnější.

Průběh ztrát (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 zmenšuje (posloupnost [konverguje](#) k nule). Nejvyšší ztráta ([limes superior](#)) dosahuje hodnoty přibližně 0,67 (kapacity intervalu), což je celkem dost velká ztráta. Jiným způsobem, jak je také možné tuto ztrátu snížit je použití základní délky dvoubajtového slova (16 bitů). To by také mohlo vést ke zrychlení a zefektivnění práce s těmito čísly na procesorech s alespoň 16-ti bitovou datovou sběrnici (což je dneska již v podstatě většina), u malých čísel by však naopak docházelo k zbytečnému plýtvání místem. 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 k úspoře místa.

I když z dnešního hlediska není žádný důvod pro snahu o úsporu místa a komentování takových věcí jako optimalizace na úrovni digitálního formátu lze spíše považovat za výsměch, jde tu také o čistotu formátu jako takového. I když svou formou je formát **BNumber** i jistou kompresí čísla, neměla by v žádném případě tato jeho vlastnost být považovaná za podstatnou. Nejdůležitější je totiž schopnost reprezentovat libovolné číslo.

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.

Další formáty (název - popis):

BNatural - přirozené číslo včetně nuly, totožné s **BNumber**

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.

Tvary čísla 1:

```
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 k 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 interpretovány takto (typ - popis):

BNatural - Nese prostě celé číslo na určeném počtu bitů

BInteger - Jednoduše je první použitelný bit (nejvyšší) znaménko. 0 - kladné, 1 - záporné a z důvodu redundance 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, nebo jiného 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 tvaru jiné délky, lze tyto konstanty takto zcela bez problému stanovit

BMultiBit - pole bitů indexovaných zprava od 0

Vhodnější bude zřejmě používání neredundantní formy.

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 - Ze základní formy **BNumber** 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. Uvedme 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. [Varianta ExtendedHighestBitsAsByteCount](#))

BInteger - Zahrnutí znaménka je již poněkud složitější operace. Je totiž nutné počítat jinak kladná a jinak záporná čísla. S ohledem na to, že cílem je 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. Uvedme 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 u procesorů s architekturou [Intel](#) jsou reálná čísla realizována pomocí standardu [IEEE 754](#), 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 interpretace:

- interpretace tečky k nejspodnějšímu bitu

```

00000000 00000000      = 64      [ (1)000000. ]

```

- interpretace tečky k nejvyššímu bitu

```

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 v podstatě zcela nevhodný. Vhodnější asi bude používat méně neobvyklou techniku přidání jedničky na konec čísla. Opět si uvedeme způsoby interpretace tečky:

- interpretace tečky k nejvyššímu bitu

```

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, kde X je báze a Y mantisa:

```

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 mínus počet násobení se poté uloží do mantisy. Tento postup bude vhodné v budoucnu co nejvíce optimalizovat (třeba by se mohla přidat do procesorů nějaká vhodná instrukční sada :-). 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 **BNatural**

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
)
```

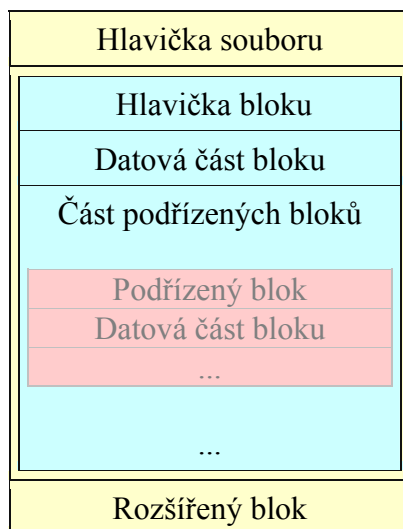
Tímto relativně jednoduchým postupem se uvolní dané konstanty posunutí významu ostatních.

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

V dalším textu budeme vždy používat právě neredundantní formy čísel.

3. Struktura formátu

Na rozdíl od vnitřní realizace čísel nelze při vytváření struktury souboru postupovat natolik logicky. Je nutné dělat kompromisy a je vhodné předpokládat, že daná řešení nebudou vždy správná a bude nutné je časem změnit. Cílem však stále zůstává najít co nejlogičtější řešení, což je věcí diskuze. Nejlepší by bylo, pokud by se nějakým způsobem podařilo prokázat logickou správnost řešení, nicméně si takový důkaz zatím nedovedu představit. Věřím, že se nakonec podaří nalézt co nejrozumější řešení, do té doby je však nutné předpokládat, že základní struktura souboru bude měněna celkem často.



Jedná se o klasickou blokovou strukturu, která se používá v mnoha současných formátech.

3.1. Hlavička souboru

Z důvodů kompatibility se současnými operačními systémy začíná každý soubor 64 bity, které umožňují rozpoznat formát XB. Jedná se o znaky "XBff" v kódu ASCII, tedy bitovou signaturou 58426666h, přičemž z důvodu rozpoznatelnosti od prostého textu jsou u znaků "f" invertovány nejvyšší bity na hodnotu E6h. Za těmito znaky následuje jeden bajt, který určuje počet datových bitů na slovo. Pro obvyklý způsob ukládání po bajtech je jeho hodnota 1111110b (FEh). Tato hlavička říká aplikaci, že soubor je strukturován na položky o 8 bitech - používané slova mají 7 datových bitů + jeden řídicí bit.

Pozn.: Je to možná trochu překvapivé, ale i hodnota 0 má v tomto případě smysl, alespoň co se čísel BNatural týče. Posloupnost bitů 1 zakončená nulou totiž představuje hodnotu určenou počtem nenulových bitů. Obecně by bylo možné řešit i další typy, jako například BInteger - sudé čísla jsou kladná, lichá záporná. Zajímavé je, že právě hodnota ByteLengthSignature je v tomto formátu.

Další hodnota, která následuje v hlavičce je **XBVersion**, která určuje základní verzi struktury bloků. Hodnota 0 byla vyhrazena pro vývojové účely. Tato hodnota již podléhá typu čísel. Celkem tedy:

DWord - FileSignature = 5842E6E6h ("XBμμ")
Byte - ByteLengthSignature = FEh (BNatural_type_0)
BNatural - XBVersion = 00h (pro účely vývoje)

Dále následují jednotlivé bloky. Pokud se v souboru nevyskytuje žádný první blok, pak se soubor nazývá prázdný. Pokud se v souboru první blok vyskytuje, určuje svým typem typ celého souboru. Za všemi bloky se nachází tzv. Rozšířený blok, který má neomezenou délku. (až na omezení operačního systému)

3.2. Blok

Blok je základní strukturální jednotkou formátu. Skládá se ze tří částí, hlavičky, datové části a části podřízených bloků.

3.2.1. Hlavička bloku

Každý blok začíná hlavičkou proměnlivé délky. Nejprve jsou dvě povinné hodnoty

BNatural - BlockGroup
BNatural - BlockType

Hodnota BlockGroup = 0 je rezervována pro systémové bloky, přičemž pokud je i BlockType = 0, jedná se o tzv. terminální blok, který již dále nepokračuje.

Pozn. Uvažoval jsem i o rezervování BG=0 pro term. blok bez BT=0, pro systémové bloky by pak byla vyhražena hodnota 1. Jiná varianta byla nedělat speciální varianty a vynutit vždy 4 hodnoty.

Kromě speciálního případu terminálního bloku obsahuje hlavička vždy ještě další dvě hodnoty:

BNatural - DataSize
BNatural - SubBlocksSize

Hodnota DataSize určuje délku datové části ve slovech. Hodnota SubBlocksSize určuje délku části obsahující podřízené objekty. Jedná se o typ BENatural (s konstantou pro nekonečné číslo), neboť kromě délky ve slovech podporuje formát ještě tzv. zero-terminated bloky, které jsou určeny právě konstantou ∞ (nekonečno). Blok pak může obsahovat blíže nespecifikovaný počet podřízených bloků ukončených terminálním blokem/kódem. Tuto variantu jsem se rozhodl zahrnout s ohledem na výpočetní náročnost složitých souborů. Bez tohoto opatření by bylo nutné vždy určovat délku podřízených bloků, což by mohl vést ke značné časové/paměťové režii. Více o zpracování souboru lze zjistit v části [Zpracování souboru](#).

Pozn. Jako další varianta přichází v úvahu uvádění počtu podřízených bloků. V případě ∞ by se nic nezměnilo, ale jinak by nebylo možné přeskočit podřízené bloky a bylo nutné je všechny zpracovat a to rekurzivně do hloubky, což si myslím může být nevýhodné (podobně jako u formátu čísel).

Hodnota BlockGroup určuje takzvanou skupinu bloků, podle toho, jak jsou rezervovány (viz. [Specifikace skupiny](#)). Hodnota 0 je vyhrazena pro základní bloky. Hodnota BlockType pak určuje konkrétní blok z dané skupiny.

3.2.2. Datová část bloku

Typ bloku také určuje strukturu datové části podle tzv. BlockDefinition (viz. [Definování vlastního bloku](#)). Datová část se obecně skládá z položek typu BNatural, které však mohou být logicky uskupovány do vyšších celků, například dvojic pro reálné čísla atp. Jedinou výjimkou je Obecný datový blok (viz. [Obecný datový blok](#)). U všech ostatních bloků platí, že mají tyto první tři hodnoty:

BNatural - Major Version
BNatural - Minor Version
BNatural - Alternative Link

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 verze je implementace [kompatibility](#).

3.2.3. Část podřízených bloků

Poslední částí bloku je prostor vyhrazený pro podřízené bloky. To umožňuje řazení bloků do stromové struktury a vyjadřování závislostí mezi bloky. Bloky jsou ukončeny terminálním blokem, nebo koncem vyhrazeného prostoru.

3.3. Zpracování souboru

Programům používajícím tento formát ukládá specifikace striktní pravidla vyhodnocování. Dělí se na pravidla pro určení platnosti a kompatibility souboru. Zatímco platnost říká, zda je soubor správně napsán, pravidla kompatibility určují, zda může být daný soubor zpracován danou aplikací.

3.3.1. Platnost souboru

Soubor je považován za platný, pokud si odpovídají velikosti bloků a jejich meze a také specifikace daných verzí a skutečné rozmístění proměnných v bloku. Aplikace má povinnost při zpracování souborů rekurzivně zkontrolovat všechny bloky ležící v cestě na jejich validitu, neboli blok je platný, pokud jsou všechny jeho nadřazené bloky platné. Například se musí kontrolovat, zda blok končí právě na konci oblasti vyhrazené pro podřízené bloky atp. Lze tak rozlišovat vlastní platnost bloku a platnost v rámci souboru. Kromě toho platí další [pravidla](#).

3.3.2. Pravidla platnosti

Blok se nazývá korektní, pokud platí:

- Datová část bloku je korektní

- to znamená, že rozměry hodnot přesně odpovídají vymezenému prostoru, tedy poslední nepřekračuje vyhrazený prostor

- Část podřízených bloků je korektní

- rozměry podřízených bloků přesně odpovídají vymezenému prostoru - žádný nepřesahuje, nebo je obsažen terminální blok, pro SubBlocksSize = ∞ (nekonečno), viz. [metoda SKIP](#)

Blok se nazývá syntakticky správný (v rámci souboru), pokud platí:

- Blok je korektní
- Všechny nadřazené bloky jsou syntakticky správné
 - je nutné rekurzivně ověřit
- Hodnoty BlockGroup a BlockType jsou platné
 - znamená to, že jsou v mezích svých intervalů určených nadřazeným [Specifikačním blokem](#)
- Počet hodnot v datovém bloku je správný
 - Specifikace bloku určuje počet hodnot, které obsahuje datová část daného bloku. Ten musí být přesně dodržen

Blok se nazývá platný (v rámci souboru a aplikace), pokud platí:

- Blok je syntakticky správný
- Nadřazené bloky jsou syntakticky správně
- Všechny jeho hodnoty jsou platné
 - patří do tzv. omezených intervalů, pokud jsou tyto definovány
- Blok je kompatibilní s aplikací
 - viz. [Pravidla kompatibility](#)

3.3.3. Pravidla kompatibility

Jak je uvedeno v části [Datová část bloku](#), obsahuje většina bloků tři hodnoty, které umožňují jednoduchou kompatibilitu. Pokud se čtený soubor liší od podporované verze pouze v hodnotě Minor Version, může aplikace soubor otevřít, a to přeskočením nepodporovaných hodnot a bloků neznámých čísel. Pokud při tomto procesu narazí na překročení meze, je vstupní soubor nekompatibilní. Pokud se soubor liší v hodnotě Major Version nesmí program soubor interpretovat a měl by nahlásit chybu, nebo aktualizovat ovladače, případně stáhnout konvertor. Musí být použita nová verze programu, nebo nabídnuty ke stažení nástroje pro konverzi. Pokud aplikace přesto není schopna daný blok interpretovat, je povinna vyhodnotit místo něj blok, na který ukazuje hodnota Alternative Link. Neplatí však žádná povinnost zpětné kompatibility a používání alternativních odkazů není vynucováno, uvádění této

hodnoty tak závisí pouze na vůli uživatele a jeho potřebě zpracovatelnosti souboru ve starších aplikacích. Interpretace významu hodnoty AlternativeLink odpovídá typu odkaz (viz. [Odkaz](#)). Pokud tento odkaz odkazuje sám na sebe (hodnota 0), je zakázáno dále zpracovávat daný blok. (Neboli hodnota 0 ukončuje vyhodnocení kompatibility) Daný blok pak není platný.

3.3.4. Specifikace verzí

Platí striktní pravidlo, že při zvýšení Minor Version musí být zachováno pořadí hodnot z předchozí verze a mohou být pouze přidávány nové hodnoty. Pokud je z nějakého důvodu vyžadována změna pořadí nebo významu již existujících hodnot, je nutné změnit hodnotu Major Version.

Kromě toho umožňuje formát specifikovat omezující podmínky pro hodnoty, aby bylo možné omezit formát pro použití na speciálních hardwarových zařízeních (viz. [Specifikace bloku](#)). Kromě toho je povinné každou specifikaci registrovat - viz. [Internetové centrum](#).

3.3.5. Vyhodnocení typu bloku

Jak již bylo uvedeno, typ bloku určují dvě hodnoty, BlockGroup a BlockType. Hodnota BlockGroup = 0 je rezervována pro systémové bloky. Vyšší hodnoty jsou určovány tzv. [Specifikačním blokem](#), a to dvěma možnými způsoby. Buď je použit specifikační blok rozšiřující, nebo překrývající. V prvním případě je k současným specifikovaným hodnotám BlockGroup přidána série dalších hodnot. V druhém případě jsou vyrušeny všechny hodnoty (kromě 0 ovšem) a místo nich je použita nová série. Tato specifikace významu platí také pro všechny podřízené bloky rekurzivně (tzn. pro všechny podřízené bloky do hloubky).

3.3.6. Doporučený postup zpracování

Postup zpracování souborů aplikací by měl být následující:

- Přečtení a ověření signatury souboru
- Určení datové jednotky

- cyklickým čtením bitů, dokud se nenarazí na jednotku je nutné určit počet datových bitů, nebo prostým ověřením hodnoty FEh, pokud program jiný typ nepodporuje

- Ověření podporované struktury bloků
- Ověření syntaktické struktury souboru

- při tomto ověření se zjišťuje, zda mají bloky uváděné velikosti a zda nepřesahují vyhrazenou oblast, nebo délku celého souboru

- Ověření kontrolních součtů

- před použitím dané větve se provedou kontrolní součty, pokud jsou uvedeny (viz. [Kontrolní součty](#))

... bude dokončeno později

3.4. Přístupové metody

I když se to na první pohled možná nezdá, je vlastně XBff programový předpis. Každý blok je zpracováván jako programový kód a chová se zhruba jako objekt. Dědičnost je realizována pomocí indexu Minor Version. Předpis každého bloku má vždy metodu READ, která vrací samotný objekt a určuje obvykle jednu přístupovou metodu GET. Rozdíl lze vidět nejlépe na bloku odkaz. Metoda READ vrací daný blok, zatímco metoda GET vrací blok na který vede odkaz.

3.4.1. Metoda GET

3.4.2. Metoda READ

3.4.3. Metoda HANDLE

Další metody: GETSIZE, SKIP, ISVALID, ISCORRECT, ISSYNTAXOK, SUBITEMCOUNT

4. Systémové bloky

Systémové bloky jsou základní bloky, které nemají obvykle svoje [definiční schéma](#). Pro tyto bloky platí, že mají BlockGroup = 0.

4.1. Obecný datový blok

Obecný datový blok je základní typ bloku, který umožňuje ukládání posloupnosti dat, hodnot o velikosti slova, tedy dle hodnoty ByteLengthSignature. Velikost části podřízených bloků je obvykle 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 vnitřně interpretovány jako pole bajtových prvků. Obecný datový blok je tedy předpis přiřazující celému číslu z intervalu $\langle 0..DataSize-1 \rangle$ číslo z intervalu $\langle 0..255 \rangle$, tento interval však závisí na používané datové šířce. Je rozumné hovořit o tomto bloku 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:

$\langle 0..DataSize-1 \rangle \rightarrow$ (obvykle: $\langle 0..255 \rangle$)

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 verzí.

4.2. Odkaz

Odkaz je blok, který umožňuje adresovat teoreticky jakoukoli datovou strukturu, v podstatě cokoli. Podstatné je však především adresování bloků uvnitř souboru, což je základní funkce. Jednotlivé bloky jsou číslovány dle pořadí, ve kterém jsou uvedeny a to tak, že hodnota 0 odkazuje na datovou část bloku samotného a vyšší indexy pak na jednotlivé podřízené bloky v pořadí v jakém jsou v souboru uloženy. Další rozšířené verze pak umožňují odkazovat se na libovolný některý z těchto bloků ve stejném nebo jiném souboru ať již v místní složce, nebo kdekoli jinde v síti, nebo internetu. V základní verzi je způsob získání 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 - PathCount
BNatural - PathIndex0
BNatural - PathIndex1
BNatural - PathIndex2
...

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. Obecně není omezena ani délka takové cesty.

- 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 aktuální jednotky "X:\", v systémech Unix se jedná o kořenovou složku "/" (root). Cesta je opět relativní:

BNatural - UpCount
BNatural - FilePathIndex
BNatural - PathCount
BNatural - PathIndex0
BNatural - PathIndex1
BNatural - PathIndex2
...

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 FilePathIndex neukazuje na soubor kompatibilní s formátem XB. Další podmínkou je, že pokud hodnota UpCount "nedosáhne" mimo vlastní soubor, musí být blok odpovídající PathIndex prázdný. Tím se dostáváme k významu hodnoty FilePathIndex, která určuje index bloku v části podřízených bloků, který je typu (nebo vrací typ) posloupnosti textových řetězců libovolného jazykového druhu. Platí také, že tento odkaz může ukazovat na soubor, nebo složku v souborovém systému. Kromě toho je zřejmě verze 0 podmnožinou verze 1. Toho je také využito u [konvertoru](#).

- 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 snad vše osvětlí:

BNatural - FileProtocolIndex
BNatural - UpCount
BNatural - PathIndex
BNatural - PathCount
BNatural - PathIndex0
BNatural - PathIndex1

BNatural - PathIndex2

...

Zpětnou "kompatibilitu" s verzí 1 nyní zajišťuje tzv. prázdný protokol, neboli protokol "internal". 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 jiný odkaz:

BNatural - LinkIndex

BNatural - UpCount

BNatural - PathIndex

BNatural - PathCount

BNatural - PathIndex0

BNatural - PathIndex1

BNatural - PathIndex2

...

Tato verze vrátí odkaz, který vznikne rozšířením odkazu z bloku adresovaného pomocí LinkIndex.

4.3. Specifikační blok

Specifikační blok slouží pro specifikaci významu hodnoty BlockGroup v hlavičce bloku. Má tyto hodnoty:

- Verze 0: BlockGroup Specifier

Tato verze vyruší všechny předchozí definice hodnot blockGroup kromě 0 a nastaví jejich nové významy.

BNatural - ItemsCount

BNatural - ItemLink0

BNatural - ItemLink1

...

Jednotlivé hodnoty ItemLink odkazují na [definiční bloky](#), které určují význam hodnoty BlockType. Je doporučeno používat odkazy s protokolem **xbfc** (viz. [Protokoly](#) a [Internetové centrum](#)).

- Verze 1: Short BlockGroup Specifier

BNatural - ItemsCount

BNatural - BlockGroupSpecLink

- Verze 2: BlockGroup Additional Specifier

Oproti předchozí verzi, tato nenahrazuje dříve specifikované hodnoty BlockGroup, jinak je totožná.

4.4. Protokol

Protokol je blok, který určuje, jakým způsobem se bude vyhodnocovat odkaz.

5. Základní řídicí bloky

XXX Tato sada bloků slouží pro ukládání některých matematických hodnot.

5.1. Určující typ

Tento blok určuje typ dat, které reprezentuje daný soubor.

5.2. Určující skupina

Určující skupina je blok, který určuje, jakým způsobem budou vyhodnocovány podřízené bloky.

Obecný datový blok je základní typ bloku, který umožňuje ukládání posloupnosti

4.2. "SubMerger"

Tento filter vrací místo ...

4.3. Protokol

4.4. Filter

4.5. Definice typu

Umožňuje definovat typ proměnných

- Verze 0: Simple
- Verze 1: Struct type

X.X. Tvůrčí entita

Slouží pro identifikaci tvůrce. Může se jednat o člověka, případně jinou myslící bytost, nebo automat.

4.2. Pole

Bloky lze dělit například podle jejich funkce, nebo výstupní hodnoty.

- datové bloky - nesou pouze konstantní data. Metoda **GET** a **READ** vrací stejnou hodnotu. Označení je skupina 1.
- filtry - bloky typu filter umožňují definovat základní prostředek pro modifikaci dat. Pokud je na něj volána metoda **GET** jsou místo vlastního obsahu vrácena zpracovaná data. To umožňuje implementaci známých funkcí, jako jsou například filtry, komprese, kódování, kryptování, slučování, kontrola chyb a další.
- předpisy - bloky tohoto typu předpisují povolené rozmezí hodnot a typů podbloků.
- skripty - určují chování jako reakci na události

Bloky těchto typů je nutné při zpracování odlišovat. Základní jsou samozřejmě datové bloky. Jedná se většinou o bloky s natolik univerzálním významem, že je není třeba vztahovat k nějakému fyzikálnímu významu, ale vystačíme zde s matematikou.

4.2. Reprezentace čísel

Pro ukládání čísel je vhodné vytvořit blok speciálního typu, neboť má číslo na rozdíl od obecného datového bloku již svoji strukturu.

4.2.1. 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

4.2.2. Časový údaj

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

- Verze 0: Reserved

- Verze 1: Basic Time

Proměnná TimeType by měla určovat například některý z typů: [Timestamp](#), [GMT](#), [DOSTime](#)

BNatural - TimeType

BReal - TimeValue

- Verze 2: Time Period

Tato verze určuje dlouhodobější časový okamžik

BNatural - TimeType

BReal - StartTimeValue

BEReal - StopTimeValue

- Verze 3: Relative Time Period

Je podobná předchozí verzi s tím rozdílem, že doba trvání je určena relativně

BNatural - TimeType

BReal - TimeValue

BEReal - PeriodLength

5.2.1. Typ časového údaje

Jak asi víte, je možné založit čas na více jednotkách, ať už je to sekunda, nebo co já vím poločas rozpadu Deuteria při absolutní nule je nutné typ rozlišit. V budoucnu snad bude možné definovat čas jako periodu kmitání kvarků, pokud bude konstantní a nezávislá na vnějších podmínkách. Do té doby ale...

- Verze 0: Reserved - pro galaktické úmluvy :-)

- Verze 1: CesiumBased Time

Založený na v současné době nejpřesnější metodě [atomových hodin](#) a frekvenci vyzářených atomů Cesia.

BReal - MultiplyConstant

BReal - StartTime

5.3 Délka

Umožňuje ukládat délku / vzdálenost

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. Nebo souřadnice v prostoru libovolného rozměru od pevně daného bodu.

5.2.1. Galaxie

Tento blok určuje konkrétní galaxii podle indexu galaxií na vesmírné centrále :-). No dobře, tak prozatím podle záznamů Observatoří >> (sehnat [link](#))

PS: Nepište mi, že moc čtu sci-fi!

- Verze 0: Reserved - rezerva pro mezigalaktickou úmluvu

- Verze 1: Basic Galaxy

BNatural - Galaxy

Galaxie podle indexu a označení z ...

- Verze 2: Galaxy string

BNatural - StringIndex

Galaxie určena podle řetězce.

5.2.2. Planeta

Tento blok určuje konkrétní galaxii podle indexu galaxií

- Verze 0: Basic

BNatural - Planet

5.3. Textový řetězec

Při ukládání textu je nutné vzít do úvahy podporu libovolného jazyka, kódování a dalších vlastností textu. Text je totiž jistá forma komprese a to jak grafických symbolů, tak významu.

5.3.1. Textová hodnota

Textová položka je vlastní vyjádření textu.

- Verze 0: Basic Text

V základní verzi obsahuje pouze odkazy na jazyk, kódování a blok vlastních textových dat.

BNatural - LanguageIndex

BNatural - EncodingIndex

BNatural - DataIndex

5.3.2. 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

5.3.3. Kódování znaků

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

- Verze 0: Reserved

- Verze 1: IANA CharacterSet Number

Využití indexů znakových sad z internetu.

BNatural - Major CharacterSet Number

BNatural - Minor CharSet Number

- Verze 2: IANA CharSet String

Tato verze využívá specifikace kódování podle jednoznačné identifikace znakovým řetězcem v kódování [ASCII](#).

BNatural - StringIndex

6. Bitmapový obrázek

Mezi informace, které jsou tradičně ukládány v binárním tvaru patří i obrazová data. Ačkoli se zdá být problém ukládání klasických bitmapových dat jednoduše řešitelný, ve formátu XB tomu tak není. Formát totiž musí reprezentovat fyzikální význam dat a být univerzální. Tyto cíle mi opravdu znesnadňují a zpomalují práci, ale doufám, že se s nimi porvu.

Základem je blok, který specifikuje, že jeho data představují bitmapový obrázek. Kromě této informace by blok neměl nést žádnou další informaci.

Prozatím nemám rozmyšlenou strukturu ani mnoho dalších věcí. Mohu tedy uvést pouze svoji aktuální představu. Soubor by měl mít zhruba tuto strukturu:

Obrázek

- Vznik - viz blok vzniku
- Plocha
 - Směšovač
 - Datové pole
 - DATA
 - Paleta
 - DATA

6.1. Pracovní rovina

Pracovní rovina je virtuální plátno, na které se bude kreslit. Základní informací by měl být rozměr plátna a objekty, které na něm jsou. Tato rovina má jistou vlastnost v každém bodě, která závisí na typu roviny.

- Verze BNatural - Link na velikost X
BNatural - Link na velikost Y
BNatural - Počet objektů
BNatural - Objekt 1
..
BNatural - Objekt N

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. Fyzikální význam je rovina která záření o frekvenci F dopadající do bodu (X,Y) pod libovolným úhlem rozdělí na tři části a to na část, která projde, část která se odrazí a část, která je pohlcena (absorbována).

- 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 jednoho stupně a výšky základní frekvence, především pomocí fyzikálních základních.

Paleta

- Verze 0: Reservováno

6.3. Bitmapového obrázek

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

7. Zvukový soubor

No řekněte, není to šílenost vyvíjet takový formát?

Pokud byste viděli již někde stejnou realizaci čísel, nebo binární formát, který má podobné cíle, tak mi napište, ať neplýtvám svým časem...

Jako uznávám, že tu toho od předchozí verze moc nepřibylo, spíš jsem to jenom přepracoval do HTML, ale co už...

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
Programový kód

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

Uzavřel návrh reprezentace čísla a domnívám se, že je v základní kostře hotov.

Dále je nutné se zabývat strukturou bloků, která není tak triviální a logická, jak se původně zdálo.

Problémy:

- jak specifikovat typ bloku?

- kolik indexů vyhradit pro určení typu bloky?

- umístit odkaz na alternativní blok?

- nutné vždy uvádět délku bloků? (režie)

- specifikace přístupových metod GET a READ, případně OPEN...

8. Podpůrné aplikace a knihovny

Ačkoli není dosud dokončen ani návrh formátu, je zároveň se specifikací vytvářena sada podpůrných aplikací a knihoven, která by uměla s formátem pracovat. Tyto nástroje jsou zatím vyvíjeny skupinou BOMI pro tyto programovací/skriptovací jazyky: Borland Pascal, Borland Delphi, Sun Java, PHP. Do budoucna se počítá s vývojem alespoň v jazyce C/C++.

Vývoj je zaměřen jednak na vývoj dynamických knihoven a jednotek/modulů a také pro vývoj samostatných nástrojů pro konverzi, zobrazení a editaci. Pro vytvoření hlavní knihovny, která by uměla zpracovávat formát XB je nutné nejprve vyvinout nástroje pro podporu neomezených čísel. Ačkoli již takovéto nástroje jistě existují, neměl jsem dosud možnost se po nich na internetu poohlédnout, takže jsem nucen vytvářet své vlastní, jistě výkonnostně nedostatečné. Následují jednotlivé projekty.

- 8.1. [Knihovna BNumLib](#)
- 8.2. [Knihovna XBLib](#)
- 8.3. [Editor formátu XB](#)

8.1. Knihovna BNumLib

Tato knihovna zapouzdřuje práci s nekonečnými čísly a jejich zpracování na a z binární podoby. Více o této podobě se můžete dozvědět v části [Neomezená čísla](#).

8.1.1. BNumLib pro Borland Delphi

Knihovna je rozdělena do několika samostatných jednotek (Unit) a zapouzdřující knihovny DLL. Jednotky jsou zaměřeny na práci s jednotlivými typy číselných formátů, např. BNatural, BInteger, BReal...

8.2. Knihovna XBLib

Tato knihovna zapouzdřuje práci s formátem XB.

8.3. Editor formátu XB

Tato aplikace umožňuje zjednodušené zpracování souboru ve formátu XB, jeho stromové prohlížení a editaci.

Referenční odkazy do sítě internet

W3C World Consorcium [<http://www.w3.org>]

XML - eXtensible Markup Language [<http://www.w3.org>]

DTD - Dynamic Tag Definiton [<http://www.w3.org>]

HTML - Hypertext Markup Language [<http://www.w3.org>]

XHTML - Extensible Hypertext Markup Language [<http://www.w3.org>]

CSS - Cascade Style Sheet [<http://www.w3.org>]

RFC - Request For Comment [<http://www.rfc.net>]

Source Forge [<http://www.sourceforge.net>]

GNU/GPL2 - General Public License Ver. 2 [<http://www.com>]

Microsoft - Microsoft © Corporation TM [<http://www.microsoft.com>]

UNICODE - Universal Code Table [<http://www.com>]

UTF - [<http://www.com>]

INTEL - Intel Corporation © [<http://www.intel.com>]

IEEE - International Electric and Electronic ? [<http://www.ieee.com>]

ISO - International Standartization Organization [<http://www.iso.com>]

ANSI - American National Standartization Organization [<http://www.ansi.com>]

GIF - CompuServe Graphic Interchange Format [<http://www.com>]

PNG - Portable Network Graphic [<http://www.com>]

Provedené úpravy

Od verze wr3 k verzi wr4:

- Pozměněn úvod
- Změna "magic number"
- Převod specifikace do UTF-8
- Změna hlavičky - přidán BlockGroup
- Přidána část 8 - BNumLib, XBLib, XBEditor
- Pravidla platnosti a kompatibility
- Změněm typ SubBlocksSize na BENatural
- Terminální blok

Připravuje se:

Množina

Množina je uskupení vzájemně různých prvků. Pokud abstrahujeme od jejich označení, lze množinu definovat počtem prvků. Tedy:

BENatural - počet prvků

Pozn.: Je lepší definovat zvlášť konečnou a nekonečnou množinu, nebo použít BENatural ?

Abeceda

Abeceda je množina dále nedělitelných prvků - znaků.

Slovo nad abecedou

Slovo je libovolná (konečná/nekonečná/prázdná) posloupnost znaků z abecedy.

Algebra

Algebra je matematická struktura, která vyznačuje množinou funkčních symbolů a množinou identit.

Automat

Konečný automat: pětice: množina stavů, vstupní abeceda, přechodová funkce, počáteční stavy, koncové stavy

Graf

Graf je množina vrcholů + množina hran

Gramatika

Gramatika je čtveřice: množiny neterminálů, terminálů(=abeceda), počáteční neterminál, a množina pravidel. Pravidlo gramatiky je dvojice slov nad množinou, která vznikne sjednocením terminálů a neterminálů. Pozn. má platit, že počáteční neterminál patří do neterminálů? (podle def. ano, ale nemusí...)

Pole/entice

Pole je n-rozměrná posloupnost prvků stejného typu s definovaným pořadím.

Typ hodnoty

3.4 Vytvoření vlastního bloku

Specifikace by měla určovat co největší počet bloků pro většinu potřeb ukládání dat. Přesto se může stát, že byste potřebovali vytvořit vlastní blok. Přes veškerou snahu k tomu pravděpodobně bude docházet často - za předpokladu, že se formát rozšíří, což ale není příliš pravděpodobné.

Základem je umístění tzv. specifikace bloku (viz. [Specifikace bloku](#)) na veřejně přístupný server, nejlépe do internetu. Pokud hodláte daný formát vážně používat, můžete požádat o registraci u [správce formátu XB](#) a bude vám přiděleno jedinečné číslo, které budete moci pro svůj formát používat.