Redbin

Poznámka: Specifikace verze 2

Redbin je binární formát, který přesně reprezentuje hodnoty Redu, uložené v paměti, přičemž umožňuje rychlé načítání (vyhýbaje se parsovací a ověřovací fázi prezentace textového formátu). Formát Redbin je převážně inspirován formátem REBin. Redbin umí zakódovat závaznou informaci pro hodnoty `any-word!`, realizovat odkazy na sdílené buffery pro hodnoty `series!` a řídit cykly pro hodnoty typu `any-block!`.

1. Kodek Redbinu

Formát Redbinu umožňuje řešit případy, typicky související se serializací dat, jako jsou:

- prostředí, založená na perzistentních stavech a zobrazeních;
- vzdálená procedurální volání;
- sdílení dat a programů v síti s jinými systémy;
- zjištění změn u časově proměnných dat.

Rozhraním, které umožňuje realizovat výše uvedené procedury a metody, je kodér a dekodér (codec) Redbinu. Tento kodek je dostupný prostřednictvím funkcí `save` a `load`, jak je popsáno níže.

Syntaxe

```
save/as <where> <value> 'redbin
save <file> <value>

load/as <data> 'redbin
load <file>

<data>  : Redbinem kódovaná hodnota typu binary!
<value> : hodnota libovolného podporovaného typu
<where> : destinace pro uložení kódovaných dat (file!, url!, string!, binary!, none!)
<file>  : soubor typu file! s extenzí .redbin
```

NOTE	Codec Redbinu neumí kódovat či dekódovat nepodporované hodnoty, potažmo hodnoty, které je obsahují.

WARNING	Attempt to `load` Redbin data with malformed payload will likely lead to unexpected results or a runtime crash.

2. Lexikální konvence

V celém tomto dokumentu jsou při popisu kódovacího formátu Redbinu používány tyto lexikální konvence:

- Čísla v oblých závorkách indikují velikost polí;
- Pole, následované rovnítkem, má pevně stanovený obsah;
- Pole, následované jménem typu záznamu v hranatých závorkách, indikuje kódovaný záznam Redbinu téhož typu;
- Tři tečky nahrazují generický záznam hodnoty libovolného typu;
- Svislá čára (pipe: |) indikuje výběr mezi alternativami;
- Znak násobení (*) indikuje opakování;
- Označení cesty se používá jako odkaz na flagy záhlaví záznamu.

3. Formát kódování

Implicitní formát kódování je optimalizován pro rychlost dekódování, zatímco *kompaktní* formát vyžaduje menší úložný prostor (za cenu mnohem pomalejšího dekódování).

NOTE	Specification of the compact encoding format is not yet defined.

Obecné rozložení dat v Redbinu je popsáno níže. Každá definice odkazuje na příslušnou část této dokumentace.

Záhlaví

Obsahuje informaci o zbývajících datech.

Tabulka symbolů

Nepovinné; je-li přítomné, obsahuje internované (interned) řetězce, používané při záznamech symbolických datových typů.

Payload

Ukládá záznamy Redbinu, které kódují hodnoty Redu.

Data v těchto sekcích jsou ukládána ve formátu *little-endian*. Všechna celočíselná pole reprezentují pozitivní (non-negative) hodnoty, avšak protože je runtime Redu interpretuje jako signované, má jejich horní limit hodnotu 2^{31}-1.

4. Záhlaví

Data v Redbinu začínají záhlavím (header), jež má následující formát:

```
magic="REDBIN" (6), version=1|2 (1), flags (1), length (4), size (4)

length : počet načítaných záznamů (root records).
size   : velikost ukládaných záznamů (payload records) v bytech.
```

Význam čísel v poli `flags` je popsán v následující tabulce.

Table 1. Redbin header flags.

Bits	Description
7-3	Rezervováno pro budoucí použití.
2	Je-li zadáno, indikuje, že data Redbinu obsahují tabulku symbolů.
1	Je-li zadáno, indikuje, že pole bezprostředně následující za polem `flags` je komprimované. Komprimační algoritmuse je nezávislý na implementaci.
0	Je-li zadáno, indikuje, že sekce záznamů je kódována v kompaktním formátu.

Záhlaví (header) je jediná povinná část kódování ve formátu Redbin; jak tabulka symbolů, tak payload lze vynechat - za předpokladu, že jsou řádně nastavená pole a flagy.

5. Tabulka symbolů

Tabulka symbolů (pokud použita) bezprostředně následuje za údaji v záhlaví. Tato tabulka je nepovinná a měla by být použita pouze tehdy, jsou-li v Redbin payload přítomny hodnoty typu `any-word!`. Tabulka symbolů má dvě části:

Tabulka offsetů

Seznam offsetů ke stringové reprezentaci symbolů uvnitř bufferu stringů;

Buffer stringů

Bezprostředně následuje za tabulkou offsetů; obsahuje spojené (concatenated), nulou ukončené

a v UTF-8 kódované řetězce. Na konci každého řetězce může být výstelka (padding) o velikosti 64 bitů.

Pozice offsetu v tabulce je dána jeho (nulou počínajícím) indexem, jenž je používán symboly jako odkaz v záznamech typu `context!` a `any-word!`. Odsazení (offsets) v tabulce jsou odstupy pojednávaných stringů v bytech od počátku sekce s buffery stringů.

Tabulka kódování offsetů je popsána níže:

```
Default: length (4), size (4), offset (4) * length
Compact: TBD
```

Pole `length` obsahuje počet vstupů v tabulce. Pole `size` indikuje velikost stringového bufferu v bytech (včetně nepovinné výstelky).

V průběhu spouštěcího (booting) runtime procesu jsou tyto symboly slučovány s tabulkou symbolů Redu a offsety jsou nahrazovány hodnotami ID symbolů z této tabulky. Runtime codec vynechává tuto slučovací fázi a invokuje symboly v místě každého relevantního dekódovaného symbolu.

Za tabulkou symbolů jsou hodnoty Redu ukládány jako sekvence záznamů bez speciálních vymezovačů (delimiters) nebo koncových markerů. Načtené hodnoty z kořenové úrovně jsou uloženy v řadách typu `block!`.

6. Definice záznamů

Každý použitelný záznam (payload) v Redbinu začíná 32 bitovým záhlavím, definovaným jako:

Table 2. Uspořádání záhlaví záznamu.

Bits	Description	Relevant datatypes
31	Flag `new-line`; je-li zadán, indikuje flag nového řádku v hodnotovém slotu.	All.
30	Flag `no-values`; je-li zadán, indikuje že záznam typu `context!` neobsahuje záznamy hodnot.	`context!`
29	Flag `stack?`; je-li zadán, indikuje že hodnoty dekódovaného záznamu typu `context!` jsou alokovány spíše ve stacku než v paměti heap.	`context!`
28	Flag `self?`; je-li zadán, indikuje že záznam typu `context!` je schopen odkázat sám na sebe prostřednictvím slova `self`.	`context!`
27-26	Pole `kind`; kóduje záznam typ `context!`.	`context!`

Bits	Description	Relevant datatypes
25	Flag `set?`; je-li zadán, indikuje že záznam typu `any-word!` je následován záznamem hodnoty, na níž dekódovaná hodota typu `any-word!` potřebuje být nastavena.	`any-word!`
24	Flag `owner?`; je-li zadán, indikuje že dekódovaná hodnota typu `object!` vlastní jednu či více hodnot.	`object!`
23	Flag `native?`; je-li zadán, indikuje že dekódovaná hodnota typu `op!` je odvozena od hodnoty typu `native!`, jinak od hodnoty typu `action!`.	`op!`
22	Flag `body?`; je-li zadán, indikuje že hodnota typu `op!` je odvozena buď od hodnoty typu `function!` nebo od hodnoty typu `routine!` a má blok s tělem funkce.	`op!`
21	Flag `complement?`; je-li zadán, indikuje že dekódovaná hodnota typu `bitset!` je komplementovaná.	`bitset!`
20	Flag `sign`; je-li zadán, indikuje že dekódovaná hodnota typu `money!` má záporné znaménko.	`money!`
19	Flag `reference?`; je-li zadán, indikuje že záznam Redbinu obsahuje odkaz.	See Reference section.
18-16	Rezervováno pro budoucí použití.	—
15-8	Pole `unit`; kóduje velikost elementu (i.e. unit) do (series) bufferu.	`series!`
7-0	Pole `type`; kóduje typ hodnoty.	All.

Dále následují individuální popisy jednotlivých typů záznamů.

6.1. Special

Některé typy záznamů Redbinu nekorespondují s žádným datovým typem Redu a jsou popsány v této sekci.

6.1.1. Padding

```
Default: header (4)
Compact: N/A

header/type=0
```

Tento prázdný záznam se používá k řádnému zarovnání (align) 64-bitových hodnot.

6.1.2. Reference

```
Default: header (4), length (4), offset (4) * length
Compact: TBD

header/type=255
```

Záznamy odkazů se používají ke kódování různých vztahů mezi hodnotami Redu, jako jsou vazby (bindings) typu `any-word!` a buffery sdílených hodnot typu `series!`.

Pole `length` určuje počet polí `offset`, obsažených uvnitř odkazového záznamu; každé pole `offset` specifikuje z nuly vycházející offset k již načtené hodnotě Redu prostřednictvím jejího rodiče, vycházejíc z kořenového bloku. Seznam takových offsetů prakticky tvoří cestu ke zmiňované hodnotě.

Hodnota Redu, jež se používá jako rodič k výpočtu offsetu, se nazývá *waypoint*; hodnota Redu, k níž je formována cesta pomocí odkazu, se nazývá *target*. Záznamy odkazů jsou obvykle používány jinými záznamy k získání datatypově specifických částí, sdílených s cílem (target). Záznam hodnoty Redu, který obsahuje odkaz (reference), se nazývá *referral*. Ve všech definicích záznamů, které následují, se formát referral používá k popisu takovéto formy kódování - jen ale je-li zadán flag `reference?` příslušného záznamu hodnoty.

Záznamy Redbinu, které mohou sloužit jako odkazy (referraly) jsou: `series!`, `map!`, `bitset!`, `any-word!`, `refinement!`, `object!`, `function!`.

Pouze vybraný počet datových typů může být waypointem neb targetem. Pravidla pro výpočet offsetu a odkazování u každého z nich jsou uvedena v následující tabulce.

Table 3. Datatypes thru and to which reference paths can be formed.

Datatypes	Waypoint	Target
`any-block!`, `map!`	Offset od čela řady. S hodnotou typu `map!` se zachází jako s lineárním blokem.	Buffer řady je opakovaně použit.
`any-string!`, `binary!`, `bitset!`, `vector!`, `image!`	—	Buffer řady je opakovaně použt.
`action!`, `native!`	Offset od čela bloku specifikací.	Spec buffer is reused.
`object!`	Offset od čela bloku hodnot.	Spojovací (binding) informace je opakovaně použita.
`any-word!`, `refinement!`	Offset do kontextu, v němž je hodnota vázána, což je prezentováno jako hodnota typu buď `object!` nebo `function!`.	Spojovací (binding) informace je opakovaně použita.

Datatypes	Waypoint	Target
`function!`	Offset o hodnotě `0` vybere blok specifikací (spec block), offset o hodnotě `1` vybere tělo bloku. Jiné hodnoty offsetů jsou zapovězeny.	Spojovací (binding) informace je opakovaně použita.
`op!`	Offset o hodnotě `0` vybere specifikační blok. Jiné hodnoty offsetu jsou zapovězeny.	Spojovací informace hodnoty typu `function!`, z níž je hodnota typu `op!` odvozena, je opakovaně použita.

Referral může cílit na svého rodiče, v kterémžto případě se tvoří cyklus.

6.2. Nepodporované

Některé datové typy hodnot (uvedených níže) nejsou formátem Redbinu podporovány.

Table 4. Red datatypes not supported by Redbin format.

Datatypes	Reason
`routine!`, `op!` derived from `routine!`	Obsahuje přímé ukazovátko (pointer) ke strojovému kódu.
`handle!`	Contains a reference to session-specific and OS-specific system resource.
`event!`	Contains a direct pointer to session-specific and OS-specific system resource.

Níže je uveden výčet dalších omezení:

- Předkompilované funkce mohou být kódovány ale při dekódování se začnou chovat jako funkce interpretované;
- V některých případech nelze kódovat klíčové slovo `self` objektu.

6.3. Datové typy

Tato část popisuje kódování záznamů v Redbinu, které korespondují s datovými typy hodnot v Redu.

6.3.1. `datatype!`

```
Default: header (4), value (4)
Compact: TBD

header/type=1
```

Pole `value` obsahuje ID datového typu, reprezentovaného jako 32-bitový integer. ==== `unset!`

```
Default: header (4)
Compact: TBD

header/type=2
```

Hodnota typu `unset!` je solitér (singleton) a lze ji kódovat jako pole `header` s ID datového typu.

6.3.2. `none!`

```
Default: header (4)
Compact: TBD

header/type=3
```

Hodnota typu `none!` je solitér (singleton) a lze ji kódovat jako pole `header` s ID datového typu.

6.3.3. `logic!`

```
Default: header (4), value=0|1 (4)
Compact: TBD

header/type=4
```

`value` content of `0` encodes a `false` value. Non-zero `value` content encodes a `true` value.

6.3.4. `block!`

```
Default:  header (4), head (4), length (4), ... * length
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=5
header/reference?=0|1
```

Pole `head` indikuje 'zero-based' odsazení pozice indexu od čela bloku. Pole `length` obsahuje počet hodnot, v bloku ukládaných. Záznamy hodnot bloku následují za polem `length`.

6.3.5. `paren!`

```
Default:  header (4), head (4), length (4), ... * length
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=6
header/reference?=0|1
```

Stejná kódovací pravidla jako `block!`.

6.3.6. `string!`

```
Default:  header (4), head (4), length (4), data (unit * length), padding (1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=7
header/unit=1|2|4
header/reference?=0|1
```

Pole `head` má stejný význam jako u ostatních záznamů řad. Pole `unit` indikuje kódovací formát řetězce; platné hodnoty jsou pouze 1, 2 a 4. Pole `length` obsahuje počet kódovacích bodů (codepoints), ukládaných v řetězci. Podporováno je až 16777215 kódpointů (2^{24} - 1). String je kódován ve formátu UCS-1, UCS-2 nebo UCS-4, v závislosti na maximální šířce obsažených kódpointů. V záznamu `dat` není přítomen žádný `nul-terminating` znak, ani není začleněn v poli `length`. Může být přítomna výstelka (padding of 1 to 3 NUL bytes) k zarovnání konce záznamu typu `string!` s 32-bitovou hranicí.

6.3.7. `file!`

```
Default:  header (4), head (4), length (4), data (unit * length), padding (1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=8
header/unit=1|2|4
header/reference?=0|1
```

Stejná kódovací pravidla jako `string!`.

6.3.8. `url!`

```
Default:  header (4), head (4), length (4), data (unit * length), padding (1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=9
header/unit=1|2|4
header/reference?=0|1
```

Stejná kódovací pravidla jako `string!`.

6.3.9. `char!`

```
Default: header (4), value (4)
Compact: TBD

header/type=10
```

Pole `value` obsahuje UCS-4 codepoint, uložený jako 32-bitový integer.

6.3.10. `integer!`

```
Default: header (4), value (4)
Compact: TBD

header/type=11
```

Pole `value` obsahuje signovaný 32-bitový integer, jenž reprezentuje kódovanou hodnotu Redu.

6.3.11. `float!`

```
Default: padding [padding], header (4), value (8)
Compact: TBD

header/type=12
```

Volitelné pole `padding` je přidáno k řádnému zarovnání pole `value` k 64-bitové hranici. Samo pole `value` obsahuje 64-bitovou IEEE 754 desetinnou číslici.

6.3.12. `context!`

```
Default: header (4), length (4), symbol (4) * length, ... * length
Compact: TBD

header/type=14
header/kind=0|1|2
header/no-values=0|1
header/stack?=0|1
header/self?=0|1
```

Kontexty jsou hodnoty Redu, interně používané některými datovými typy, jako je `function!`, `object!` a odvozené typy. Záznam kontextu obsahuje dva za sebou jdoucí seznamy. První je seznam zadaných slov v kontextu, reprezentovaných jako `symbolické` odkazy. Druhý obsahuje asociované záznamy hodnot pro každý symbol z prvního seznamu.

Pole `kind` v záhlaví záznamu kóduje typ kontextu: `0` pro globální kontext, `1` pro kontext funkce a `2` pro kontext objektu. Globální kontext není nikdy kódován explicitně, což znamená, že jsou použity pouze hodnoty `1` a `2`. Pole `length` indikuje počet zápisů v kontextu.

Je-li zadán flag `no-values`, znamená to, že za symboly nejsou žádné záznamy hodnot (prázdný obsah). Je-li zadán flag `stack?`, potom jsou hodnoty alokovány ve stacku místo v paměti heap. Flag `self?` se používá k indikaci toho, že obsah může ošetřit na sebe odkazující slovo (`self` v objektech).

6.3.13. `word!`

```
Default:  header (4), symbol (4), index (4), ...|context [object!|function!]
Referral: header (4), symbol (4), index (4), context [reference]
Compact:  TBD

header/type=15
header/set?=0|1
header/reference?=0|1
```

Pole `symbol` je index v tabulce symbolů Redbinu. Termín `index` je index slova v kontextu, k němuž je slovo vázáno. Je-li zadán flag `set?`, potom je slovo vázáno ke globálnímu kontextu a pole `index` je následováno záznamem hodnoty, na niž má být slovo nastaveno. Není-li zadán flag `set?`, je pole `index` následováno záznamem typu `object!` nebo `function!`, jenž obsahuje kontext, k němuž má být slovo vázáno.

NOTE V aktuální implementaci zadaný flag `set?` indikuje, že je slovo vázáno ke globálnímu kontextu ale záznam hodnoty je vynechán.

6.3.14. `set-word!`

```
Default:  header (4), symbol (4), index (4), ...|context [object!|function!]
Referral: header (4), symbol (4), index (4), context [reference]
Compact:  TBD

header/type=16
header/set?=0|1
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `word!`.

6.3.15. `lit-word!`

```
Default:  header (4), symbol (4), index (4), ...|context [object!|function!]
Referral: header (4), symbol (4), index (4), context [reference]
Compact:  TBD

header/type=17
header/set?=0|1
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `word!`.

6.3.16. `get-word!`

```
Default:  header (4), symbol (4), index (4), ...|context [object!|function!]
Referral: header (4), symbol (4), index (4), context [reference]
Compact:  TBD

header/type=18
header/set?=0|1
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `word!`.

6.3.17. `refinement!`

```
Default:  header (4), symbol (4), index (4), ...|context [object!|function!]
Referral: header (4), symbol (4), index (4), context [reference]
Compact:  TBD

header/type=19
header/set?=0|1
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `word!`.

6.3.18. issue!

```
Default: header (4), symbol (4)
Compact: TBD

header/type=20
```

Pole `symbol` je index v tabulce symbolů Redbinu.

6.3.19. native!

```
Default: header (4), ID (4), spec [block!]
Compact: TBD

header/type=21
```

`ID` field is an offset into the internal `natives/table` jump table, followed by a `block!` record encoding native's spec.

6.3.20. action!

```
Default: header (4), ID (4), spec [block!]
Compact: TBD

header/type=22
```

`ID` field is an offset into the internal `actions/table` jump table, followed by a `block!` record encoding action's spec.

6.3.21. op!

```
Default: header (4), parent [function!]|spec [block!], ID (4)
Compact: TBD

header/type=23
header/body?=0|1
neader/native?=0|1
```

Zadaný flag `body?` indikuje, že hodnota typu `op!` je odvozena z hodnoty typu `function!`. Není-li flag `body?` zadán, potom je hodnota typu `op!` odvozena buď z typu `action!` nebo `native!` — volba mezi oběmi možnostmi je indikována flagem `native?`.

Je-li zadán flag `body?`, potom je pole `header` následováno záznamem typu `function!`, který kóduje rodiče hodnoty typu `op!`. Jinak je následováno záznamem typu `block!`, který kóduje specifikaci hodnoty typu `op!` a rovněž `ID` hodnoty buď typu `action!` nebo `native!`.

6.3.22. function!

```
Default:  header (4), spec-size (4), body-size (4), context [context!], spec [block!],
body [block!]
Referral: header (4), context [reference]
Compact:  TBD

header/type=24
header/reference?=0|1
```

Položky spec-size a body-size určují velikosti bloků spec a body a jsou použity dekodérem pro předalokaci.

Cílem odkazu (reference) je hodnota typu function!, op! nebo any-word!. Hodnota typu function! (načtená hodnota, rodič hodnoty typu op! nebo kontext hodnoty typu any-word!) je kopírována slovo od slova, což znamená, že referral sdílí nejenom vázací informaci ale také blok specifikací a blok těla.

6.3.23. path!

```
Default:  header (4), head (4), length (4), ... * length
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=25
header/reference?=0|1
```

Stejná kódovací pravidla jako u hodnoty typu block!.

6.3.24. lit-path!

```
Default:  header (4), head (4), length (4), ... * length
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=26
header/reference?=0|1
```

Stejná kódovací pravidla jako u hodnoty typu block!.

6.3.25. set-path!

```
Default:  header (4), head (4), length (4), ... * length
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=27
header/reference?=0|1
```

Stejná kódovací pravidla jako u hodnoty typu `block!`.

6.3.26. `get-path!`

```
Default:  header (4), head (4), length (4), ... * length
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=28
header/reference?=0|1
```

Stejná kódovací pravidla jako u hodnoty typu `block!`.

6.3.27. `bitset!`

```
Default:  header (4), length (4), data (length), padding (1-3)
Referral: header (4), buffer [reference]
Compact:  TBD

header/type=30
header/complement?=0|1
```

Zadaný flag `complement?` indikuje, že je bitset komplementován. Pole `length` kóduje počet uložených bajtů. Pole `data` je paměťové uložiště (dump) pro buffer řad typu `bitset!`, pořadí bajtů je zachováno. Pole `data` potřebuje být obloženo (padded) dostatečným počtem nulových bajtů aby mohl být následující záznam zarovnán s 32-bitovou hranici.

6.3.28. `object!`

```
Default:  header (4), class (4), on-set (4), arity (4), context [context!]
Referral: header (4), context [reference]
Compact:  TBD

header/type=32
header/owner?=0|1
header/reference?=0|1
```

Pole `class` uchovává ID třídy objektu. Pole `on-set` je dvojice 16-bitových celých čísel, kódujících

offset k funkcím `on-change*` a `on-deep-change*` v bloku hodnot objektu. Pole `arity` má stejný formát jako `on-set` avšak kóduje arity jednotlivých funkcí. Tato dvě pole jsou volitelná a jsou kódována pouze tehdy, je-li zadán flag `owner?` v záhlaví záznamu.

6.3.29. `typeset!`

```
Default: header (4), array1 (4), array2 (4), array3 (4)
Compact: TBD

header/type=33
```

Pole `array1`, `array2` a `array3` tvoří bitset, v němž index každého bitu `1` indikuje ID datového typu, obsaženého v typesetu.

6.3.30. `error!`

```
Default: header (4), code (4), ... * 6
Compact: TBD

header/type=34
```

Pole `code` kóduje identifikátor chyby a je následováno šesti záznamy hodnot pro pole chyby: `arg1`, `arg2`, `arg3`, `near`, `where`, `stack`.

6.3.31. `vector!`

```
Default:  header (4), head (4), length (4), type (4), data (unit * length), padding
(1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=35
header/unit=1|2|4|8
```

Pole `type` obsahuje ID datového typu elementu vektoru. Pole `unit` indikuje velikost jeho typu v bajtech. Jsou podporovány pouze tyto kombinace hodnot `type` a `unit`:

Table 5. Combinations of `vector!` fields.

Type	Unit
`char!`, `integer!`	1, 2, 4
`float!`	4, 8
`percent!`	8

Pole `data` obsahuje seznam hodnot. Je-li pole `unit` rovno 1 či 2, musí být pole `data` doplněno

nulovými bajty až 32-bitové hranici.

6.3.32. `pair!`

```
Default: header (4), x (4), y (4)
Compact: TBD

header/type=37
```

Pole `x` a `y` kódují jednotlivé elementy páru jako 32-bitová celá čísla.

6.3.33. `percent!`

```
Default: padding [padding], header (4), value (8)
Compact: TBD

header/type=38
```

Stejná pravidla kódování jako u typu `float!`.

6.3.34. `tuple!`

```
Default: header (4), array1 (4), array2 (4), array3 (4)
Compact: TBD

header/type=39
header/unit=3-12
```

Pole `unit` kóduje velikost entice (tuple) v bajtech; jsou povoleny pouze hodnoty od `3` do `12`. Pole `array1`, `array2` a `array3` tvoří dohromady paměťové uložiště pro payload slotu entice.

6.3.35. `map!`

```
Default:  header (4), length (4), ... * length
Referral: header (4), buffer [reference]
Compact:  TBD

header/type=40
header/reference?=0|1
```

Pole `length` obsahuje počet kódovaných elementů (klíčů i hodnot).

6.3.36. `binary!`

```
Default:  header (4), head (4), length (4), data (length)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=41
header/reference?=0|1
```

Pole `data` obsahuje paměťové uložiště pro buffer binárních řad (binary's series); pořadí bajtů je zachováno.

6.3.37. `time!`

```
Default: padding [padding], header (4), value (8)
Compact: TBD

header/type=43
```

Stejná pravidla kódování jako u typu `float!`.

6.3.38. `tag!`

```
Default:  header (4), head (4), length (4), data (unit * length), padding (1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=44
header/unit=1|2|4
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `string!`.

6.3.39. `email!`

```
Default:  header (4), head (4), length (4), data (unit * length), padding (1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=45
header/unit=1|2|4
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `string!`.

6.3.40. `date!`

```
Default: header (4), date (4), time (8)
Compact: TBD

header/type=47
```

Pole `date` obsahuje datovou hodnotu, vloženou do 32-bitového celého čísla. Používá se následující formát (velikosti polí jsou v bitech):

```
year (15), time? (1), month (4), day (5), timezone (7)
```

Sub-pole `year` a `timezone` obsahují signované hodnoty. Pole `time` ukládá časovou hodnotu jako 64-bitový float.

6.3.41. `money!`

```
Default: header (4), currency (1), amount (11)
Compact: TBD

header/type=49
header/sign=0|1
```

Pole <code>amount</code> je sekvence ždibců (nibbles), reprezentujících bázi (17) a subjednotku (5) peněžní hodnoty; pořadí bajtů je zachováno. Je-li zadán flag <code>sign</code>, je částka interpretována jako negativní. Pole <code>currency</code> je celočíselná hodnota, reprezentující ID měny (0 pro generické peníze, ≤ 255 pro kód existující měny).

6.3.42. `ref!`

```
Default:  header (4), head (4), length (4), data (unit * length), padding (1-3)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=50
header/unit=1|2|4
header/reference?=0|1
```

Stejná pravidla kódování jako u typu `string!`.

6.3.43. `image!`

```
Default:  header (4), head (4), length (4), rgba (4 * width * height)
Referral: header (4), head (4), buffer [reference]
Compact:  TBD

header/type=51
header/reference?=0|1
```

Pole `length` je dvojice 16-bitových celých čísel, kódujících šířku a výšku zobrazení. Pole `rgba` obsahuje RGBA obsah obrázku (4 bytes per pixel) se zachovaným pořadím bajtů.