

# Rich Text Dialect

## Table of Contents

1. Úvod .....	1
2. High-level RT dialekt .....	1
3. Low-level RT dialekt .....	3
4. Face rich-text .....	4
4.1. Single-box mode .....	4
4.2. Multi-box mode .....	5
5. Funkce, těžící informace .....	5

## 1. Úvod

'Rich text' (RT) je bohatší alternativa k 'plain text'. Podporuje formátování textu **bold**, italics, podtržení, dále podporuje různé fonty, jejich velikosti a barevné provedení.

Programovací rozhraní (API) pro RT má tři úrovně, od nejjednodušší k nejvíce optimalizované:

- Použití RTD z prostředí VID (nebo při manuálním vytváření piškotu).
- Low-level RTD pro jeden řetězec (kvůli rychlosti).
- Vícero RT odstavců v jediném piškotu (pro složitější uspořádání).

## 2. High-level RT dialekt

### Gramatika

```

nested: [ahead block! into rtd]
color: [
  s: tuple! (v: s/1) ;-- color as R.G.B tuple
  | issue! (v: hex-to-rgb s/1) ;-- color as #rgb or #rrggbb hex value
  | word! if (tuple? attempt [v: get s/1])
]
f-args: [
  ahead block! into [integer! string! | string! integer!]
  | integer!
  | string!
]
style!: make typeset! [word! tag! tuple! path!]
style: [ahead style! [
  ['b | 'bold | <b>] (push 'b) [nested | rtd [/b | /bold | </b>]] (pop
'b)
  | ['i | 'italic | <i>] (push 'i) [nested | rtd [/i | /italic | </i>]] (pop
'i)
  | ['u | 'underline | <u>] (push 'u) [nested | rtd [/u | /underline | </u>]] (pop
'u)
  | ['s | 'strike | <s>] (push 's) [nested | rtd [/s | /strike | </s>]] (pop
's)
  | ['f | 'font | <font>]
    s: f-args (push either block? s/1 [head insert copy s/1 'f][reduce ['f s/1]])
    [nested | rtd [/f | /font | </font>]]
    (pop 'f)
  | ['bg | <bg>] color (push reduce ['bg v]) [nested | rtd [/bg | </bg>]] (pop 'bg)
  | color (push-color v) opt [nested (pop-color)]
  | ahead path!
  into [
    (col: 0 insert/only mark tail stack) some [ ;@@ implement any-single
      (v: none)
      s: ['b | 'i | 'u | 's | word! if (tuple? attempt [v: get s/1])]
      (either v [col: col + 1 push-color v][push s/1])
    ](insert cols col)
  ]
  nested (pop-all take mark)
]]
rtd: [some [pos: style | s: [string! | char!] (append text s/1 s-idx: tail-idx?)]]

```

## POZNÁMKA

Skladba cesty (path) umožňuje kombinovat více stylů, následně aplikovaných na blok za cestou. Smíšené použití oddělovačů a bloků pro různé styly je povoleno.

## Použití

Vstup RTD je zpracován specifickou funkcí `rtd-layout`, která vrací piškot (face) `single-box rich-text`, v němž je kód RTD kompilován na jediný textový řetězec, uložený v aspektu (facet) `/text` a `low-level` popis stylu, uložený v aspektu `/data`.

Úplnou specifikací funkce je:

```
rtd-layout func [  
    "Returns a rich-text face from a RTD source code"  
    spec [block!] "RTD source code"  
    /only "Returns only [text data] facets"  
    /with "Populate an existing face object"  
        face [object!] "Face object to populate"  
    return: [object! block!]  
]
```

Příklad:

```
rt: rtd-layout [<i> <b> "Hello" </b> <font> 24 red " Red " </font> blue "World!" </i>]  
  
view [rich-text with [text: rt/text data: rt/data]]
```

RTD lze přímo poskytnout ve VID pro aspekt **data**.

Příklady:

```
view compose [rich-text 200x100 data [i b "Hello" /b font 24 red " Red " /font blue  
"World!" /i]]  
  
view compose [rich-text 200x100 data [i [b ["Hello"] red font 24 [" Red "] blue  
"World!"]]]  
  
view compose [rich-text 200x100 data [i/b/u/red ["Hello" font 32 " Red " /font blue  
"World!"]]]  
  
view compose [rich-text 200x100 data [i/blue ["Hello " b/u/red [font [32 "Arial"] "Red  
" /font] "World!"]]]
```

### 3. Low-level RT dialekt

Tento dialekt popisuje seznam stylů, použitelných na řetězec, uváděný v aspektu **/text** piškotu **rich-text**. Účelem tohoto dialektu je poskytnout co nejrychlejší řešení pro dynamické změny a evokaci (querying) informací. Dialekt je v dobré shodě s přítomnými hardwarově urychlovanými API (ve Windows se opírá o Direct2D).

#### Použití

Gramatikou dialektu je jednoduchý seznam textových segmentů (definovaných počáteční pozicí a délkou, vyjádřených jako hodnota pair!), následovaný seznamem stylů. Takže, jeho typická struktura je:

```
[
  <range1> style1 style2 ...      ;-- range1: start1 x length1
  <range2> style1 style2 ...      ;-- range2: start2 x length2
  ...
]
```

Styly se mohou překrývat a pozdější styl má vyšší prioritu (kaskádující styly).

Jsou podporovány tyto styly:

```
[
  tuple!                                ;-- text color
  | backdrop tuple!                     ;-- background color
  | bold                                ;-- bold font
  | italic
  | underline tuple! (color) lit-word! ('dash, 'double, 'triple) ;@@ color and type
are not supported yet
  | strike tuple! (color) lit-word! ('wave, 'double)                ;@@ color and type
are not supported yet
  | border tuple! (color) lit-word! ('dash, 'wave)                  ;@@ not
implemented
  | integer!                                ;-- new font size
  | string!                                ;-- new font name
]
```

#### POZNÁMKA

Barva textu nemá následovat bezprostředně za dekorátorem **strike** nebo **underline**. Modifikace barvy a typu za uvedenými pokyny by se vztahovala na úpravu čáry a nikoli textu. Poněvadž tyto dekorátory nejsou ještě zavedeny, nemá určení barvy nebo typu za těmito klíčovými slovy žádný účinek.

## 4. Face rich-text

Nový nativní píškot (face) **rich-text** podporuje vlastnosti bohatého textu s hardwarovým urychlením. Tento píškot má dva režimy zobrazení bohatého textu.

### 4.1. Single-box mode

Pro zobrazení bohatého textu je použita celá plocha píškotu, počínajíc v levém horním rohu s použitím následujících aspektů (facets):

- `/data (block!)`: blok low-level instrukcí, jež mají být použity v aspektu text.
- `/text (string!)`: textový řetězec, zobrazený s použitím instrukcí v aspektu `/data`.

Případně použitý aspekt **draw** bude vykreslený nad zobrazením rich textu.

Příklady:

```
view [  
  rich-text with [  
    text: "Hello Red World!"  
    data: [1x17 0.0.255 italic 7x3 255.0.0 bold 24 underline]  
  ]  
]  
view [  
  rich-text "Hello Red World!"  
  with [data: [1x17 0.0.255 italic 7x3 255.0.0 bold 24 underline]]  
]
```

## 4.2. Multi-box mode

V tomto režimu lze uvnitř téhož RT piškotu zobrazit libovolné množství RT oblastí. Za tím účelem je v dialektu Draw každá RT oblast označena klíčovým slovem.

Specifické aspekty (facets):

- `/draw (block!)`: blok textových instrukcí, případně proložených regulárními instrukcemi Draw.
- `/text (none!)`: pro umožnění tohoto režimu musí tento aspekt nastaven na `none`.

### Rozšíření Draw

```
text <pos> <text>
```

`<pos>` : hodnota typu `pair!`, udávající levý horní roh textového boxu  
`<text>` : řetězec nebo piškot `rich-text` s RT popisem v jediném boxu

Příklad:

```
view compose/deep [  
  rich-text 200x200 draw [  
    text 10x10 (rt1: rtd-layout ["Some^/" b "text^/" /b "here"] rt1/size: 50x80  
rt1)  
    text 100x90 (rt2: rtd-layout [red "Other^/" b "text^/" /b "there"] rt1/size:  
50x80 rt2)  
    pen gold box 90x80 160x180  
  ]  
]
```

## 5. Funkce, těžící informace

Pro získávání informací o obsahu RT piškotu jsou k dispozici následující funkce. Tyto funkce se

použijí k usnadnění:

- navigace kurzoru
- hit testing

Z implicitního kontextu `system/words`:

```
caret-to-offset: function [  
    "Given a text position, returns the corresponding coordinate relative to the top-  
    left of the layout box"  
    face    [object!]  
    pos     [integer!]  
    return: [pair!]  
]  
  
offset-to-caret: function [  
    "Given a coordinate, returns the corresponding text position"  
    face    [object!]  
    pt      [pair!]  
    return: [integer!]  
]  
  
size-text: function [  
    "Returns the area size of the text in a face"  
    face [object!]  
    /with                                ;-- unused for rich-text  
        text [string!]  
    return: [pair! none!]  
]
```

Z rich-text kontextu:

```
line-height?: function [  
    "Given a text position, returns the corresponding line's height"  
    face    [object!]  
    pos     [integer!]  
    return: [integer!]  
]  
  
line-count?: function [  
    "number of lines (> 1 if line wrapped)"  
    face    [object!]  
    return: [integer!]  
]
```

Příklady:

```

view [
  rich-text data [font 16 "Select some text with your mouse" /font]
  on-down [
    bkg: reduce [ ; Background for selected text
      as-pair caret: offset-to-caret face event/offset 0
      'backdrop sky
    ]
    either 2 = length? face/data [ ; On first selection
      pos: tail face/data
      append face/data bkg
    ][ ; Changing starting pos on subsequent selections
      change pos bkg/1
    ]
  ] all-over
  on-over [
    if event/down? [ ; On dragging change only length
      pos/1/2: (offset-to-caret face event/offset) - caret
    ]
  ]
]

```

```

view compose/deep [
  rich-text draw [
    text 10x10 (rt: rtd-layout [i/blue ["Hello " red/b [font 24 "Red " /font]
    "World!"]])
    line-width 5 pen gold
    line ; Let's draw line under words using a pair of above helper functions
      (as-pair 10 h: 10 + rich-text/line-height? rt 1) ; Starting-point y -> 10
    + line-height
      (as-pair 10 + pick size-text rt 1 h) ; End-point x -> 10 + length-of-text-
    size
  ]
]

```