Jeux de types (Typesets)

Table of Contents

1. Résumé . . . 1
2. Jeux de types disponibles en Red . . . 1
   - 2.1. all-word! . . . 1
   - 2.2. any-block! . . . 2
   - 2.3. any-function! . . . 2
   - 2.4. any-list! . . . 2
   - 2.5. any-object! . . . 2
   - 2.6. any-path! . . . 2
   - 2.7. any-string! . . . 2
   - 2.8. any-type! . . . 2
   - 2.9. any-word! . . . 2
   - 2.10. default! . . . 2
   - 2.11. external! . . . 3
   - 2.12. immediate! . . . 3
   - 2.13. internal! . . . 3
   - 2.14. number! . . . 3
   - 2.15. scalar! . . . 3
   - 2.16. series! . . . 3

1. Résumé

Les jeux de types représentent des ensembles de valeurs de type `datatype!` stockées dans un tableau de bits compact (jusqu'à 96 bits), permettant une haute performance dans la vérification de type en temps réel.

Les types de données dans un jeu de types peuvent partager des traits ou comportements communs, mais cela n'est pas requis. Un jeu de types peut être créé à partir de tous critères correspondant aux besoins de l'utilisateur.

Voir: typeset!

2. Jeux de types disponibles en Red

2.1. all-word!

- make typeset! [word! set-word! lit-word! get-word! refinement! issue!]

2.2. any-block!

- make typeset! [block! paren! path! lit-path! set-path! get-path! hash!]

2.3. any-function!

- make typeset! [native! action! op! function! routine!]

2.4. any-list!

- make typeset! [block! paren! hash!]

2.5. any-object!

- make typeset! [object! error!]

2.6. any-path!

- make typeset! [path! lit-path! set-path! get-path!]

2.7. any-string!

- make typeset! [string! file! url! tag! email!]

2.8. any-type!

- make typeset! [datatype! unset! none! logic! block! paren! string! file! url! char! integer! float! word! set-word! lit-word! get-word! refinement! issue! native! action! op! function! path! lit-path! set-path! get-path! routine! bitset! object! typeset! error! vector! hash! pair! percent! tuple! map! binary! time! tag! email! handle! date! image! event!]

2.9. any-word!

- make typeset! [word! set-word! lit-word! get-word!]

2.10. default!

- make typeset! [datatype! none! logic! block! paren! string! file! url! char! integer! float! word! set-word! lit-word! get-word! refinement! issue! native! action! op! function! path! lit-path! set-path! get-path! routine! bitset! object! typeset! error! vector! hash! pair! percent! tuple! map! binary! time! tag! email! handle! date! image! event!]

2.11. external!

- make typeset! [event!]

2.12. immediate!

- make typeset! [datatype! none! logic! char! integer! float! word! set-word! lit-word! get-word! refinement! issue! typeset! pair! percent! tuple! time! handle! date!]

2.13. internal!

- make typeset! [unset! float! percent!]

2.14. number!

- make typeset! [integer! float! percent!]

2.15. scalar!

- make typeset! [char! integer! float! pair! percent! tuple! time! date!]

2.16. series!

- make typeset! [block! paren! string! file! url! path! lit-path! set-path! get-path! vector! hash! binary! tag! email! image!]

  Une série en Red est définie comme une séquence d'éléments, et une position de départ qui peut être déplacée le long de la séquence à partir de la première position (`head`), jusqu'à la dernière position (`tail`). La position de départ d'une série vide est la dernière position (`tail`).

  Plusieurs références peuvent être faites à la même séries avec des position de départ différentes:

```
>> a: "hello"
== "hello"

>> b: next a
== "ello"

>> index? a
== 1

>> index? b
== 2

>> same? a b
== false

>> same? a head b
== true

>> append a " world"
== "hello world"

>> b
== "ello world"
```

Le type des éléments dans une série dépend du datatype! de la série. par exemple, une série de type block! peut contenir des valeurs de n'importe quel type. Une série de type string! ne peut contenir que des valeurs de type char!, etc.

`Series!` fournit une variable index qui peut être exploitée par toutes les valeurs d'action! sur les séries.