Formal Specification of Cicero by Hyeok Kim

Supplementary Material to a CHI 2022 paper, "Cicero: A Declarative Grammar for Responsive Visualization"

Notations	Examples	Description
":=": is defined as a tuple of	`a := b, c, d`	a is defined as a tuple of b, c, and d.
"~": possible names for ... are ...	`a ~ b, c`	The possible names of a are b and c.
"|": alternate argument	`a | b | c`	Either one of a, b, or c.
"...": extensible arguments	`a := b | c | ...`	a can be either b, c, or something else.
"<>": datatype	`A<String>`	A is a string type argument.
"[]": a list of	`<String>[]`	A list of string elements.
	`<a, b>[]`	A list of tuples composed of a and b.
"?": optional argument	`a?`	a is optional.
	`...?`	optional additional arguments
"{}": key-value map (e.g., JavaScript Object)	`{ <A> : <B> }`	An object with a type A key and type B value.
"()": An omittable part of a string	`(a.)b`	"a." can be omitted.

Notes

"<Number>" either a number or a string of a number with its unit (e.g., 350, "350px").

Formal Specification

```
CiceroSpec := Name?, Metadata?, Transformations

Name := <String>

Metadata := Condition?, MediaType?, AspectRatio?, ...?
Condition := xsmall | small | medium | large | xlarge | ...
MediaType := screen | paper | ...
AspectRatio := portrait | landscape | <Number> | ...

Transformations := <Rule>[]
Rule := Specifier, Action, Option?

Specifier := Role, Mark?, Index?, Id?, Data?, Field?, Values?, Datatype?,
             Structure query                Data query
    Channel?, Operation?, Interaction?, $OtherAttributes?
    Attribute query                      $OtherAttributes include encoding channels, role values, and other
                                         appearance-related properties (e.g., font styles, stroke styles, etc.).

Role := data | (data.)transform | view | (view.)row | (view.)column | (view.)facet | (view.)axis
        data   data transformation(s)  chart view   row (y) / column (x) encodings  chart facets   axes
      | (view.)hAxis | (view.)vAxis | (view.)axis.grid | (view.)axis.domain | (view.)axis.tick
        horizontal axes   vertical axes   axis grids   axis domain(s)   axis tick(s)
      | (view.)axis.label | (view.)axis.title | (view.)hAxis.grid | (view.)hAxis.domain
        axis labels   axis title   horizontal axis grids   horizontal axis domain(s)
      | (view.)hAxis.tick | (view.)hAxis.label | (view.)axis.title | (view.)vAxis.grid
        horizontal axis ticks   horizontal axis label(s)   horizontal axis title   vertical axis grids
      | (view.)vAxis.domain | (view.)vAxis.tick | (view.)vAxis.label | (view.)vAxis.title
        vertical axis domain(s)   vertical axis ticks   vertical axis label(s)   vertical axis title
      | (view.)layer | (view.)layer.transform | (view.layer.)mark | (view.layer.)mark.label
        layer(s)   layer-wise data transformation(s)   mark(s)   mark label(s)
      | (view.layer.mark.)tooltip | (view.layer.)legend | (view.layer.)legend.title
                   tooltip(s)   legend(s)   legend title
      | (view.layer.)legend.label | (view.layer.)legend.mark | (view.)title | (view.)annotation
        legend label(s)   legend mark(s)   chart title(s)   (non-data) annotations
      | (view.)emphasis
        (non-data) emphases
```

```
Mark := point | circle | rect | bar | line | ...
```
For information, read https://vega.github.io/vega-lite/docs/mark.html
```
Index := <Number> | first | last | even | odd
Id := <String>

Data := Datum | <Datum>[]
Datum := { <Field>: (<Any> | <Any>[] | <Op>[]) }
Op:= { <Operator>: <Any> }
Operator := not | and | or | == | > | >= | startsWith | ...
Field := <String>
Values := <Any>[]
Datatype := nominal | ordinal | quantitative | temporal ...

Channel := x | y | color | fill | stroke | opacity | fillOpacity | strokeOpacity | strokeDash | size
    | shape | arc | radius | theta | ...
Operation := OperationType | <OperationType>[]
OperationType := filter | aggregate | bin | ...
Interaction := InteractionType | <InteractionType>[]
InteractionType := zoom | context | ...

$OtherAttributes ~ position, x, y, dx, dy, color, fill, stroke, opacity, shape, size, width, height,
    strokeWidth, strokeDash, label, title, fontColor, fontSize, bin, aggregate, scale, translate, ...
$OtherAttributes := <Any> | By | Prod
By := <Number>
```
Addition to an existing value
```
Prod := <Number>
```
Product with an existing value
```
Action := modify | reposition | transpose | add | duplicate | remove | replace | swap

Option := Specifier | To?, From?
To := Specifier
From := Specifier
```

Role semantics in Option

When an Option has a <Specifier> form, then Role is optional. If Role is not provided, then the Role is automatically that of the specifier. If provided, the Role of the Option should be subordinate to that of the specifier, and the other properties in the Option is considered as the elements specified by the Option's role keyword. If a Role keyword B can be directly concatenated after A with the dot operator, then B is subordinate to A (e.g., A.B). However, Role is required for To and/or From, and their Roles are considered independent of the Specifier (i.e., not subordinate to the Specifier).

Examples

```
{specifier: {
  role: layer,
  ... },
  action: ...,
  option: {
    role: mark,
    ... }}
```
The Option's role: layers' mark(s).

```
{specifier: {
  role: view,
  ... },
  action: ...,
  option: {
    role: mark,
    ... }}
```
The Option's role is not valid because view.mark is not possible.

```
{specifier: {
  role: legend,
  ... },
  action: ...,
  option: {
    to: { axis.label }
    ... }}
```
The Option's To's role: axis label(s), irrelevant to the legend(s) specified by the specifier.