Input Masking

with

Vue

@shortdiv

Divya Sasidharan

Developer Advocate

0.00

0.01

0.12

1.23

12.34

123.45

1,234.56

123.45

12.34

1.23

0.12

0.01

0.00

$0


$
1

$12

$123

$1,234

$1,234.5

$1,234^{56}$

$1,234.5

$1,234

$123

$12


$
1

$0

I don't have any UX research to cite, but *anecdotally*, **I like it** when inputs that expect data in a specific format use an input mask.

– *Chris Coyier*

Why use
this pattern?

Input Events

The Role of Reactivity

Reusable component patterns

0.00

Requirements

Allow only numbers

Add format for dollars and cents


Input Events

Patterns

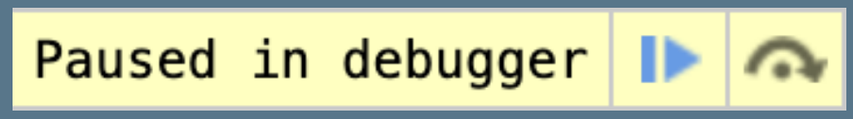
Capturing Key Events

Patterns

Capturing Key Events

Patterns

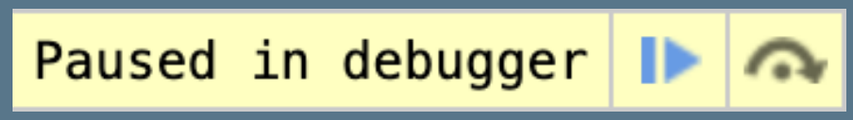
Capturing Key Events

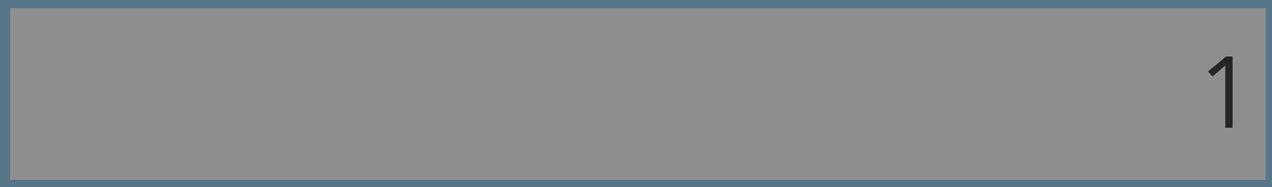


1

Patterns

Capturing Key Events

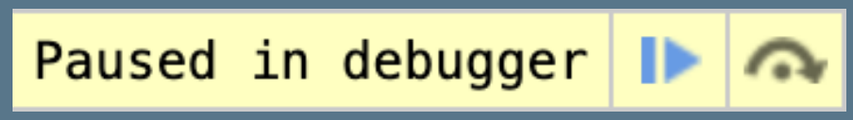




`oninput`

Patterns

Capturing Key Events



1

```
@input
```

```
<input
  type="text"
  id=“currency"
  placeholder="0.00"
  @input="formatCashMoney"
  :value="formattedCashMoney"
/>
```

```
<input
  type="text"
  id=“currency"
  placeholder="0.00"
  @input="formatCashMoney"
  :value="formattedCashMoney"
/>
```

```
<input
  type="text"
  id=“currency"
  placeholder="0.00”
  v-model="formattedCashMoney"
/>
```

```vue
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
  />
</template>

<script>
  export default {
    name: “v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
  }
</script>
```

```vue
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    }
  }
</script>
```

```vue
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id=“currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id=“currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

Requirements

Allow only numbers

Add format for dollars and cents

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {
        //format cashMoney
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {
        var key = e.key
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e){{
        var key = e.key
        if (!/^\d+/g.test(key)) {
          e.preventDefault()
        }
      }
    }
  }
</script>
```

Working Demo

keydown > keypress > input > keyup

```
<template>
  <input
    type="text"
    id=“currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {
        var key = e.key
        if (!/^\d+/g.test(key)) {
          e.preventDefault()
        }
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {
        var key = e.key
        if (!/^\d+/g.test(key)) {
          e.preventDefault()
        }
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {
        var key = e.key
        if (!/^\d+/g.test(key)) {
          e.preventDefault()
        }
      }
    }
  }
</script>
```

Working Demo

Requirements

~~Allow only numbers~~

Add format for dollars and cents

```
<template>
  <input
    type="text"
    id=“currency"
    :value="formattedCashMoney"
    @keydown="formatCashMoney"
  />
</template>

<script>
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {
        var key = e.key
        if (!/^\d+/g.test(key)) {
          e.preventDefault()
        }
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="formatCashMoney"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {

        …

        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, ",")
}


function decimalize(numstr) {
  if (numstr.length > 1) {
    let dp = numstr.substring(numstr.length-2, numstr.length)
    let denom = numstr.substring(0, numstr.length - 2)
    if (denom.substring(0,1) === "0") {
      denom = denom.substring(1, denom.length)
    }
    return `${denom}.${dp}`
  } else {
    return `0.0${numstr}`
  }
}


function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, “,”)
}


function decimalize(numstr) {
  if (numstr.length > 1) {
    let dp = numstr.substring(numstr.length-2, numstr.length)
    let denom = numstr.substring(0, numstr.length - 2)
    if (denom.substring(0,1) === "0") {
      denom = denom.substring(1, denom.length)
    }
    return `${denom}.${dp}`
  } else {
    return `0.0${numstr}`
  }
}


function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, “,”)
}


function decimalize(numstr) {
  if (numstr.length > 1) {







  } else {

  }
}



function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, “,”)
}


function decimalize(numstr) {
  if (numstr.length > 1) {
    let dp = numstr.substring(numstr.length-2, numstr.length)
    let denom = numstr.substring(0, numstr.length - 2)
    if (denom.substring(0,1) === "0") {
      denom = denom.substring(1, denom.length)
    }
    return `${denom}.${dp}`
  } else {
    return `0.0${numstr}`
  }
}


function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, “,”)
}


function decimalize(numstr) {
  if (numstr.length > 1) {
    let dp = numstr.substring(numstr.length-2, numstr.length)
    let denom = numstr.substring(0, numstr.length - 2)
    if (denom.substring(0,1) === "0") {
      denom = denom.substring(1, denom.length)
    }
    return `${denom}.${dp}`
  } else {
    return `0.0${numstr}`
  }
}


function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}
```

```
function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, ",")
}
```

thousandths

123456

cents

thousandths

1,234.56

cents

Working Demo

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="formatCashMoney"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney() {

        …

        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

keydown > keypress > input > keyup

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="formatCashMoney"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {

        …

        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="formatCashMoney"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      formatCashMoney(e) {

        …

        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      validateInput(e) { … }
      formatInput(e) {
        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      validateInput(e) { … }
      formatInput(e) {
        this.formattedCashMoney = wearMask(e.target.value)
      }
    }c
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      validateInput(e) { … }
      formatInput(e) {
        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

```vue
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      validateInput(e) { … }
      formatInput(e) {
        let unformatted = removeMask(e.target.value)
        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, “,”)
}


function decimalize(numstr) {
  if (numstr.length > 1) {
    let dp = numstr.substring(numstr.length-2, numstr.length)
    let denom = numstr.substring(0, numstr.length - 2)
    if (denom.substring(0,1) === "0") {
      denom = denom.substring(1, denom.length)
    }
    return `${denom}.${dp}`
  } else {
    return `0.0${numstr}`
  }
}


function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}

function removeMask(str) {
  return numstr.replace(/\D+/g, "");
}
```

```
function wearMask (str) {
  let num = decimalize(str)
  num = separateByThousandths(num, “,”)
}


function decimalize(numstr) {
  if (numstr.length > 1) {
    let dp = numstr.substring(numstr.length-2, numstr.length)
    let denom = numstr.substring(0, numstr.length - 2)
    if (denom.substring(0,1) === "0") {
      denom = denom.substring(1, denom.length)
    }
    return `${denom}.${dp}`
  } else {
    return `0.0${numstr}`
  }
}



function separateByThousandths(str, delimiter) {
  return str.replace(/\B(?=(\d{3})+(?!\d))/g, `${delimiter}`);
}

function removeMask(str) {
  return numstr.replace(/\D+/g, "");
}
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      validateInput(e) { … }
      formatInput(e) {
        let unformatted = removeMask(e.target.value)
        this.formattedCashMoney = wearMask(e.target.value)
      }
    }
  }
</script>
```

Requirements

~~Allow only numbers~~

~~Add format for dollars and cents~~

The Role of Reactivity

0.00

Methods

```
methods: {
  onInput() {},
  onKeyPress() {},
  onKeyDown() {}
  …
}
```

```
<input
  type="text"
  id="currency"
  :value="formattedCashMoney"
  @input="formatInput"
  @keydown="validateInput"
/>
```

```
<input
  type="text"
  id="currency"
  :value="formattedCashMoney"
  @input="formatInput"
  @keydown="validateInput"
/>
```

Reactivity

Computed Property

```
computed: {
  currentMask() {
    // format //
  }
}
```

```
<input
  type="text"
  id="currency"
  :value="formattedCashMoney"
  @input="formatInput"
  @keydown="validateInput"
/>
```

```vue
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formattedCashMoney: null
      }
    },
    methods: {
      validateInput(e) { … }
      formatInput(e) {
        let unformattedVal = removeMask(e.target.value)
        this.formattedCashMoney = wearMask(unformattedVal)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="updateInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        unformatted: null
      }
    },
    methods: {
      validateInput(e) { … }
      updateInput(e) {
        this.unformatted = removeMask(e.target.value)
      }
    },
    computed: {
      formattedCashMoney() {
        return wearMask(this.unformatted)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        unformatted: null
      }
    },
    methods: {
      validateInput(e) { … }
      updateInput(e) {
        this.unformatted = removeMask(e.target.value)
      }
    },
    computed: {
      formattedCashMoney() {
        return wearMask(this.unformatted)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="formatInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        unformatted: null
      }
    },
    methods: {
      validateInput(e) { … }
      updateInput(e) {
        this.unformatted = removeMask(e.target.value)
      }
    },
    computed: {
      formattedCashMoney() {
        return wearMask(this.unformatted)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @keydown="validateInput"
    @input="updateInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        unformatted: null
      }
    },
    methods: {
      validateInput(e) { … }
      updateInput(e) {
        this.unformatted = removeMask(e.target.value)
      }
    },
    computed: {
      formattedCashMoney(){
        return wearMask(this.unformatted)
      }
    }
  }
</script>
```

Working Demo

Reactivity

Computed Property

```
computed: {
  currentMask() {
    get(){ … }
    set(){ … }
  }
}
```

```
<template>
  <input
    type="text"
    id="currency"
    :value="formattedCashMoney"
    @input="updateInput"
    @keydown="validateInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        unformatted: null
      }
    },
    methods: {
      validateInput(e) { … }
      updateInput(e) {
        this.unformatted = removeMask(e.target.value)
      }
    },
    computed: {
      formattedCashMoney() {
        return wearMask(this.unformatted)
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
    @keydown="validateInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formatted: null
      }
    },
    methods: {
      validateInput(e) { … }
    },
    computed: {
      formattedCashMoney() {
        get() { … },
        set() { … }
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
    @keydown="validateInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formatted: null
      }
    },
    methods: {
      validateInput(e) { … }
    },
    computed: {
      formattedCashMoney() {
        get() { return this.formatted },
        set() { … }
      }
    }
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-model="formattedCashMoney"
    @keydown="validateInput"
  />
</template>

<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero",
    data () {
      return {
        formatted: null
      }
    },
    methods: {
      validateInput(e) { … }
    },
    computed: {
      formattedCashMoney() {
        get() { return this.formatted },
        set(newVal) {
          let val = removeMask(newVal)
          this.formatted = wearMask(this.unformatted)
        }
      }
    }
  }
</script>
```

Working Demo

Reusable Components

0.00

(###) ### ####

MM/DD/YYYY

<VDinero />

Component Pattern

Renderless Components

```
export default {
  render(h) {
    return
      this.$scopedSlots.default({
        …
      })
  }
}
```

```
<v-mask>
  //some children nodes//
</v-mask>
```

```
<v-mask
  v-slot:default="{ … }"
>
  //some children nodes//
</v-mask>
```

```
<v-mask
  v-slot:default="{
    formattedValue, input, keydown
  }"
>
  <label>
    <input
      type="text"
      :value="formattedValue"
      @input="input"
      @keydown="keydown"
    />
  </label>
</v-mask>
```

```
<v-mask
  v-slot:default="{
    formattedValue, input, keydown
  }"
>
  <label>
    <input
      type="text"
      :value="formattedValue"
      @input="input"
      @keydown="keydown"
    />
  <label>
</v-mask>
```

```
<v-mask
  v-slot:default="{
    formattedValue, input, keydown
  }"
>
  <label>
    <input
      type="text"
      :value="formattedValue"
      @input="input"
      @keydown="keydown"
    />
  <label>
</v-mask>
```

```
<v-mask
  v-slot:default="{
    formattedValue, listeners
  }"
>
  <label>
    <input
      type="text"
      :value="formattedValue"
      v-on="listeners"
    />
  <label>
</v-mask>
```

```
<v-mask
  v-slot:default="{
    formattedValue, listeners
  }"
>
  <label>
    <input
      type="text"
      :value="formattedValue"
      v-on="listeners"
    />
  <label>
</v-mask>
```

```
export default {
  name: "v-mask",
  data() {
    return {
      formatted: "0.00"
    };
  },
  render() {
    return this.$scopedSlots.default({
      formattedValue: this.formatted,
      listeners: {
        keydown: e => { … },
        input: e => { … }
      }
    })
  }
}
```

```
export default {
  name: "v-mask",
  data() {
    return {
      formatted: "0.00"
    };
  },
  render() {
    return this.$scopedSlots.default({
      formattedValue: this.formatted,
      listeners: {
        keydown: e => { … },
        input: e => { … }
      }
    })
  }
}
```

```
export default {
  name: "v-mask",
  data() {
    return {
      formatted: "0.00"
    };
  },
  render() {
    return this.$scopedSlots.default({
      formattedValue: this.formatted,
      listeners: {
        keydown: e => { … },
        input: e => { … }
      }
    })
  }
}
```

```
export default {
  name: "v-mask",
  data() {
    return {
      formatted: “0.00"
    };
  },
  render() {
    return this.$scopedSlots.default({
      formattedValue: this.formatted,
      listeners: {
        keydown: e => { … },
        input: e => { … }
      }
    })
  }
}
```

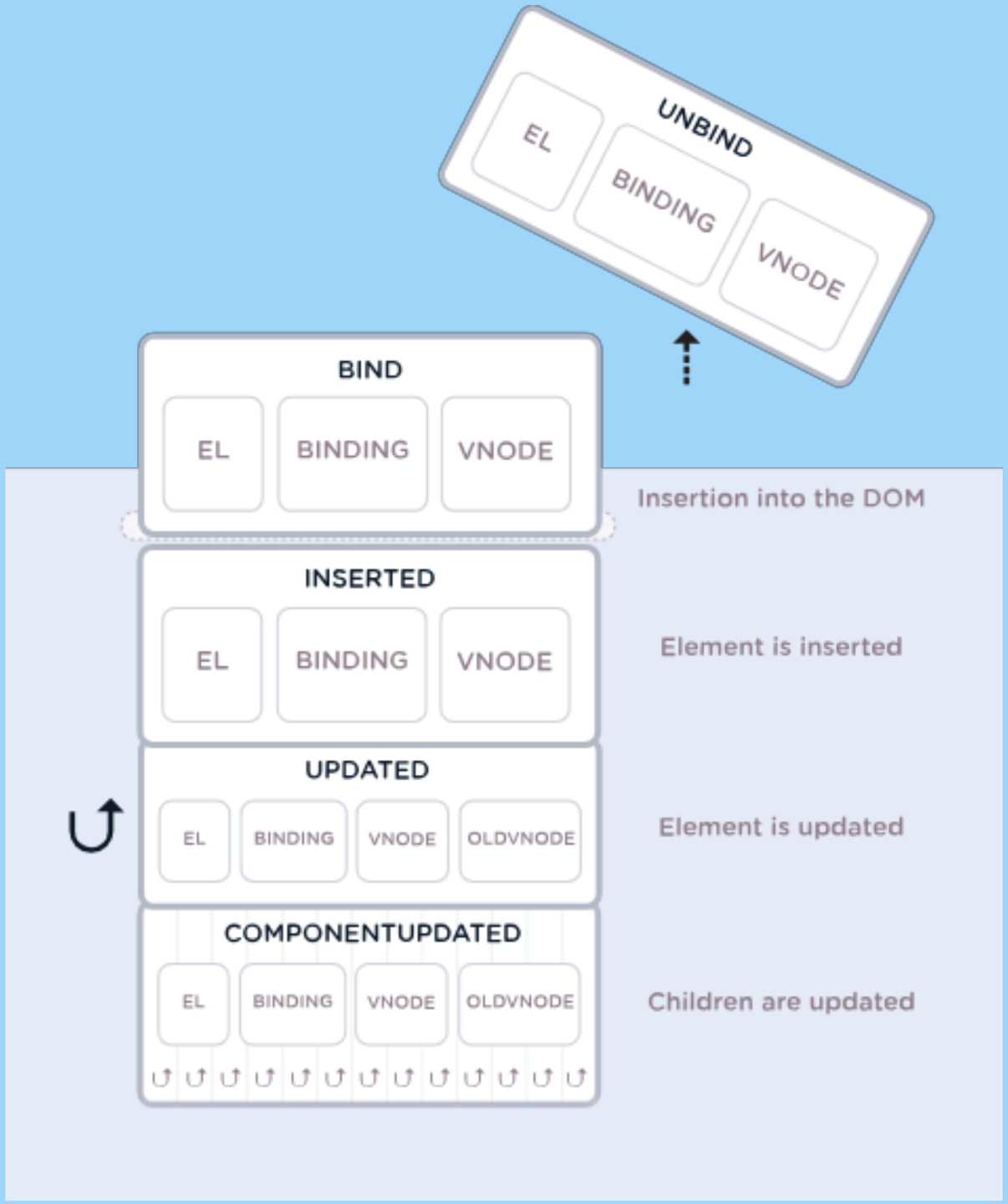
```
<input
  v-mask="'(###)###-####'"
/>
```

```
<input
  v-mask="'##/##/####'"
/>
```

Component Pattern

Directives

```
directives: {
   mask: {
      bind(){ … }
      inserted(){ … }
      update(){ … }
      componentUpdated(){ … }
      unbind(){ … }
   }
}
```


UNBIND
EL
BINDING
VNODE
BIND
EL
BINDING
VNODE
Insertion into the DOM
INSERTED
EL
BINDING
VNODE
Element is inserted
UPDATED
EL
BINDING
VNODE
OLDVNODE
Element is updated
COMPONENTUPDATED
EL
BINDING
VNODE
OLDVNODE
Children are updated

```
<template>
  <input
    type="text"
    id="currency"
    v-mask="'(###)###-####'"
  >
</template>
<script>
  export default {
    name: "v-dinero",
    directives: {
      mask: {
        bind() {…}
        componentUpdated() {…}
      }
    },
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-mask="'(###)###-####'"
  >
</template>
<script>
  export default {
    name: "v-dinero",
    directives: {
      mask: {
        bind() {…}
        componentUpdated() {…}
      }
    },
  }
</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-mask="'(###)###-####'"
  >
</template>
<script>
  import { wearMask, removeMask } from “./utils"
  export default {
    name: "v-dinero",
    directives: {
      mask: {
        bind(el, { value }) {
          const mask = value
          el.value = wearMask(el.target.value, mask);
        },
        componentUpdated() {…}
      }
    },
  }


</script>
```

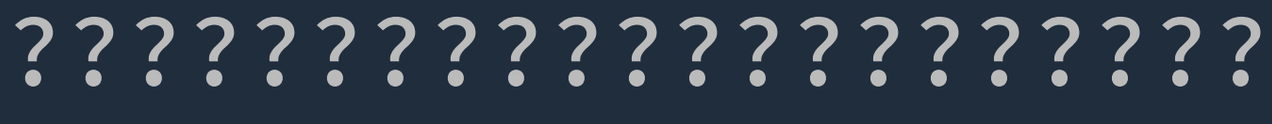
```
<template>
  <input
    type="text"
    id="currency"
    v-mask="'(###)###-####'"
  >
</template>
<script>
  import { wearMask, removeMask } from “./utils"
  export default {
    name: "v-dinero",
    directives: {
      mask: {
        bind(el, { value }) {
          const mask = value
          el.value = wearMask(el.target.value, mask);
        },
        componentUpdated(el, { value }) {
          const mask = value
          let unformatted = removeMask(el.target.value)
          el.value = wearMask(unformatted, mask);
        }
      }
    },
  }


</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-mask="'(###)###-####'"
    v-model="currentMask"
  >
</template>
<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero
    data() {
      return { currentMask: null }
    },
    directives: {
      mask: {
        bind(el, { value }) {
          const mask = value
          el.value = wearMask(el.target.value, mask);
        },
        componentUpdated(el, { value }) {
          const mask = value
          let unformatted = removeMask(el.target.value)
          el.value = wearMask(unformatted, mask);
        }
      }
    },
  }

</script>
```

```
<template>
  <input
    type="text"
    id="currency"
    v-mask="'(###)###-####'"
    v-model="currentMask"
  >
</template>
<script>
  import { wearMask, removeMask } from "./utils"
  export default {
    name: "v-dinero
    data() {
      return { currentMask: null }
    },
    directives: {
      mask: {
        bind(el, { value }) {
          const mask = value
          el.value = wearMask(el.target.value, mask);
        },
        componentUpdated(el, { value }) {
          const mask = value
          let unformatted = removeMask(el.target.value)
          el.value = wearMask(unformatted, mask);
        }
      }
    },
  }

</script>
```

Component Pattern

Hooks



Component Pattern

“Hooks”

??????????????????????

Component Pattern

~~"Hooks"~~

Compositional Functions

???????????????????????

Mixins **can't consume and use state from one to another**, but Hooks can. This means that *if we need to chain encapsulated logic*, it's now possible with Hooks.

– *Evan You*

Why use this pattern?

yyx990803 / vue-hooks

Watch 36 | Star 1,257 | Fork 34

Code | Issues 6 | Pull requests 0 | Projects 0 | Wiki | Insights

Experimental React hooks implementation in Vue

27 commits | 1 branch | 0 releases | 4 contributors

Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

yyx990803 0.3.2 — Latest commit 5e0de4e on Jan 23

.gitignore	give effects access to this when appropriate	2 months ago
README.md	readme	5 months ago
example.html	Removes dynamic imports for Firefox support	5 months ago
example2.html	Removes dynamic imports for Firefox support	5 months ago
example3.html	Removes dynamic imports for Firefox support	5 months ago
index.js	give effects access to this when appropriate	2 months ago
package.json	0.3.2	2 months ago

README.md

vue-hooks

POC for using React Hooks in Vue. Totally experimental, don't use this in production.

```
<input
  type="text"
  v-model="formatted"
  @keydown="validateInput"
>
```

```
<input
  type="text"
  :value="formatted.get()"
  @input=“formatted.set(e)"
  @keydown=“validateInput"
>
```

```
import { useData, useComputed } from "vue-hooks";

export function computeMask() {
  function wearMask(numstr) {
    //format code//
  }

  const data = useData({
    unformatted: null
  });

  const computed = useComputed(() => {
    return {
      formatted: {
        get() {
          return data.unformatted;
        },
        set(e) {
          data.unformatted = wearMask(e.target.value);
        }
      }
    }
  }
  return {
    computed
  };
}
```

```
import { useData, useComputed } from "vue-hooks";

export function computeMask() {
  function wearMask(numstr) {
  //format code//
  }

  const data = useData({
    unformatted: null
  });

  const computed = useComputed(() => {
    return {
      formatted: {
        get() {
          return data.unformatted;
        },
        set(e) {
          data.unformatted = wearMask(e.target.value);
        }
      }
    }
  }
  return {
    computed
  };
}
```

```
import { useData, useComputed } from "vue-hooks";

export function computeMask() {
  function wearMask(numstr) {
  //format code//
  }

  const data = useData({
    unformatted: null
  });

  const computed = useComputed(() => {
    return {
      formatted: {
        get() {
          return data.unformatted;
        },
        set(e) {
          data.unformatted = wearMask(e.target.value);
        }
      }
    }
  })
  return {
    computed
  };
}
```

```
import { useData, useComputed } from "vue-hooks";

export function computeMask() {
  function wearMask(numstr) {
    //format code//
  }

  const data = useData({
    unformatted: null
  });

  const computed = useComputed(() => {
    return {
      formatted: {
        get() {
          return data.unformatted;
        },
        set(e) {
          data.unformatted = wearMask(e.target.value);
        }
      }
    }
  }
  return {
    computed
  };
}
```

```
import { useData, useComputed } from "vue-hooks";

export function computeMask() {
  function wearMask(numstr) {
    //format code//
  }

  const data = useData({
    unformatted: null
  });

  const computed = useComputed(() => {
    return {
      formatted: {
        get() {
          return data.unformatted;
        },
        set(e) {
          data.unformatted = wearMask(e.target.value);
        }
      }
    }
  }
  return {
    computed
  };
}
```

```vue
<template>
  <input type="text"
    :value="formatted.get()"
    @input="formatted.set(e)"
  />
</template>

<script>
  import { computeMask } from "../hooks/computeMask"
  export default {
    name: "VDinero",
    hooks() {
      const { computed } = computeMask();
      const { formatted } = computed

      return {
        formatted
      };
    }
  }
</script>
```

```vue
<template>
  <input type="text"
    :value="formatted.get()"
    @input="formatted.set(e)"
  />
</template>

<script>
  import { computeMask } from "../hooks/computeMask"
  export default {
    name: "VDinero",
    hooks() {
      const { computed } = computeMask();
      const { formatted } = computed

      return {
        formatted
      };
    }
  }
</script>
```

```vue
<template>
  <input type="text"
    :value="formatted.get()"
    @input="formatted.set(e)"
  />
</template>

<script>
  import { computeMask } from "../hooks/computeMask"
  export default {
    name: "VDinero",
    hooks() {
      const { computed } = computeMask();
      const { formatted } = computed

      return {
        formatted
      };
    }
  }
</script>
```

```vue
<template>
  <input type="text"
    :value="formatted.get()"
    @input="formatted.set(e)"
  />
</template>

<script>
  import { computeMask } from "../hooks/computeMask"
  export default {
    name: "VDinero",
    hooks() {
      const { computed } = computeMask();
      const { formatted } = computed

      return {
        formatted
      };
    }
  }
</script>
```

Maybe don't use that...

– Vue people I chatted with

Why use
this pattern?

Advanced Reactivity API #22

Open **yyx990803** wants to merge 2 commits into `master` from `advanced-reactivity-api`

yyx990803 commented 21 hours ago — Member

Rendered

API for creating and observing standalone reactive values outside components.

```
import { state, value, computed, watch } from '@vue/observer'

// reactive object
// equivalent of 2.x Vue.observable()
const obj = state({ a: 1 })

// watch with a getter function
watch(() => obj.a, value => {
  console.log(`obj.a is: ${value}`)
})

// a "ref" object that has a .value property
const count = value(0)

// computed "ref" with a read-only .value property
const plusOne = computed(() => count.value + 1)

// refs can be watched directly
watch(count, (count, oldCount) => {
  console.log(`count is: ${count}`)
})
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```
import { value, computed } from '@vue/observer'

export function useComputeMask() {
  const unformatted = value(0)

  const formatted = computed(
    () => unformatted.value,
    val => {
      let unformattedVal = removeMask(val)
      unformatted.value = wearMask(unformattedVal)
    }
  )

  return {
    formatted
  }
}
```

```vue
<template>
  <input type="text"
    v-model="formatted"
  />
</template>

<script>
  import { useComputemask } from "../cp"
  export default {
    name: "VDinero",
    data() {
      const { formatted } = useComputemask();

      return {
        formatted
      };
    }
  }
</script>
```

Advanced Reactivity API #22

Open **yyx990803** wants to merge 2 commits into `master` from `advanced-reactivity-api`

yyx990803 commented 21 hours ago — Member

Rendered

API for creating and observing standalone reactive values outside components.

```
import { state, value, computed, watch } from '@vue/observer'

// reactive object
// equivalent of 2.x Vue.observable()
const obj = state({ a: 1 })

// watch with a getter function
watch(() => obj.a, value => {
  console.log(`obj.a is: ${value}`)
})

// a "ref" object that has a .value property
const count = value(0)

// computed "ref" with a read-only .value property
const plusOne = computed(() => count.value + 1)

// refs can be watched directly
watch(count, (count, oldCount) => {
  console.log(`count is: ${count}`)
})
```

Input Events

The Role of Reactivity

Reusable component patterns

Methods, Computeds, Directives, Renderless Components, ~~Hooks~~ Compositional Functions, Oh my!

Usability
User Experience
Developer Ergonomics

Thanks!

@shortdiv