Inputmask

Inputmask is a javascript library which creates an input mask. Inputmask can run against vanilla javascript, jQuery and jqlite.

An inputmask helps the user with the input by ensuring a predefined format. This can be useful for dates, numerics, phone numbers, ...

Highlights:

- easy to use
- optional parts anywere in the mask
- possibility to define aliases which hide complexity
- date / datetime masks
- numeric masks
- lots of callbacks
- non-greedy masks
- many features can be enabled/disabled/configured by options
- supports readonly/disabled/dir="rtl" attributes
- support data-inputmask attribute(s)
- alternator-mask
- regex-mask
- dynamic-mask
- preprocessing-mask
- JIT-masking
- value formatting / validating without input element
- AMD/CommonJS support
- dependencyLibs: vanilla javascript, jQuery, jqlite
- Android support

Demo page see http://robinherbots.github.io/Inputmask

Thanks to Jetbrains for providing a free license for their excellent Webstorm IDE.

Setup

dependencyLibs

Inputmask can run against different javascript libraries. You can choose between:

- inputmask.dependencyLib (vanilla)
- inputmask.dependencyLib.jquery
- inputmask.dependencyLib.jqlite

- .... (others are welcome)

Classic web with <script> tag

Include the js-files which you can find in the `dist` folder.
Inputmask with jQuery as dependencylib.

```
<script src="jquery.js"></script>
<script src="dist/jquery.inputmask.js"></script>
```

Inputmask with vanilla dependencylib.

```
<script src="dist/inputmask.js"></script>
```

If you like to automatically bind the inputmask to the inputs marked with the data-inputmask- ... attributes you may also want to include the inputmask.binding.js

```
<script src="dist/bindings/inputmask.binding.js"></script>
```

webpack

Install the package

```
npm install inputmask --save
```

Install the latest beta version

```
npm install inputmask@next --save
```

In your modules

If you want to include the Inputmask and all extensions.

```
var Inputmask = require('inputmask');

//es6
import Inputmask from "inputmask";
```

For individual extensions. Every extension exports the Inputmask, so you only need to import the extensions. See example.

```
require("inputmask/lib/extensions/inputmask.numeric.extensions");
var Inputmask = require("inputmask/lib/extensions/inputmask.date.extensions");

//es6
import "inputmask/lib/extensions/inputmask.numeric.extensions";
import Inputmask from "inputmask/lib/extensions/inputmask.date.extensions";
```

Selecting the dependencyLib

By default the vanilla dependencyLib is used. You can select another dependency by creating an alias in the webpack.config.

```
 resolve: {
        alias: {
            "./dependencyLibs/inputmask.dependencyLib": "./dependencyLibs/inputmask.dependencyLib.jquery"
        }
    },
```

Usage

via Inputmask class

```
var selector = document.getElementById("selector");

var im = new Inputmask("99-9999999");
im.mask(selector);

//or

Inputmask({"mask": "(999) 999-9999", .... other options .....}).mask(selector);
Inputmask("9-a{1,3}9{1,3}").mask(selector);
Inputmask("9", { repeat: 10 }).mask(selector);

Inputmask({ regex: "\\d*" }).mask(selector);
Inputmask({ regex: String.raw`\d*` }).mask(selector);
```

via jquery plugin

```
$(document).ready(function(){
  $(selector).inputmask("99-9999999");  //static mask
  $(selector).inputmask({"mask": "(999) 999-9999"}); //specifying options
  $(selector).inputmask("9-a{1,3}9{1,3}"); //mask with dynamic syntax
});
```

via data-inputmask attribute

```
<input data-inputmask="'alias': 'datetime'" />
<input data-inputmask="'mask': '9', 'repeat': 10, 'greedy' : false" />
<input data-inputmask="'mask': '99-9999999'" />

$(document).ready(function(){
  $(":input").inputmask();
  or
  Inputmask().mask(document.querySelectorAll("input"));
});
```

Any option can also be passed through the use of a data attribute. Use data-inputmask-<*the name of the option*>="value"

```
<input id="example1" data-inputmask-clearmaskonlostfocus="false" />
<input id="example2" data-inputmask-regex="[a-za-zA-Z0-9!#$% '*+/=?^_`{|}~-]+(?:\.[a-zA-Z0-
9!#$% '*+/=?^_`{|}~-]+)*@(?:[a-zA-Z0-9](?:[a-zA-Z0-9-]*[a-zA-Z0-9])?\.)+[a-zA-Z0-9](?:[a-zA-Z0-9-]*[a-zA-
Z0-9])?" />

$(document).ready(function(){
  $("#example1").inputmask("99-9999999");
  $("#example2").inputmask();
});
```

Allowed HTML-elements

- `<input type="text">`
- `<input type="search">`
- `<input type="tel">`
- `<input type="url">`
- `<input type="password">`
- `<div contenteditable="true">` (and all others supported by contenteditable)
- `<textarea>`
- any html-element (mask text content or set maskedvalue with jQuery.val)

The allowed input types are defined in the supportsInputType option. Also see (input-type-ref)

Default masking definitions

- 9 : numeric
- a : alphabetical
- * : alphanumeric

There are more definitions defined within the extensions.
You can find info within the js-files or by further exploring the options.

Masking types

Static masks

These are the very basic of masking. The mask is defined and will not change during the input.

```
$(document).ready(function(){
  $(selector).inputmask("aa-9999");  //static mask
  $(selector).inputmask({mask: "aa-9999"});  //static mask
});
```

Optional masks

It is possible to define some parts in the mask as optional. This is done by using [ ].

Example:

```
$('#test').inputmask('(99) 9999[9]-9999');
```

This mask wil allow input like `(99) 99999-9999` or `(99) 9999-9999`.
Input => 12123451234 mask => (12) 12345-1234 (trigger complete)
Input => 121234-1234 mask => (12) 1234-1234 (trigger complete)
Input => 1212341234 mask => (12) 12341-234_ (trigger incomplete)

skipOptionalPartCharacter

As an extra there is another configurable character which is used to skip an optional part in the mask.

```
skipOptionalPartCharacter: " "
```

Input => 121234 1234 mask => (12) 1234-1234 (trigger complete)

When `clearMaskOnLostFocus: true` is set in the options (default), the mask will clear out the optional part when it is not filled in and this only in case the optional part is at the end of the mask.
For example, given:

```
$('#test').inputmask('999[-AAA]');
```

While the field has focus and is blank, users will see the full mask ___-___. When the required part of the mask is filled and the field loses focus, the user will see `123`. When both the required and optional parts of the mask are filled out and the field loses focus, the user will see `123-ABC`.

Optional masks with greedy false

When defining an optional mask together with the greedy: false option, the inputmask will show the smallest possible mask as input first.

```
$(selector).inputmask({ mask: "9[-9999]", greedy: false });
```

The initial mask shown will be "_" instead of "_-____".

Dynamic masks

Dynamic masks can change during the input. To define a dynamic part use { }.

{n} => n repeats {n|j} => n repeats, with j jitmasking {n,m} => from n to m repeats {n,m|j} => from n to m repeats, with j jitmasking

Also {+} and {*} is allowed. + start from 1 and * start from 0.

```
$(document).ready(function(){
  $(selector).inputmask("aa-9{4}");  //static mask with dynamic syntax
  $(selector).inputmask("aa-9{1,4}");  //dynamic mask ~ the 9 def can be occur 1 to 4 times

  //email mask
  $(selector).inputmask({
    mask: "*{1,20}[.*{1,20}][.*{1,20}][.*{1,20}]@*{1,20}[.*{2,6}][.*{1,2}]",
    greedy: false,
    onBeforePaste: function (pastedValue, opts) {
      pastedValue = pastedValue.toLowerCase();
      return pastedValue.replace("mailto:", "");
    },
    definitions: {
      '*': {
        validator: "[0-9A-Za-z!#$%&'*+/=?^_`{|}~\-]",
        casing: "lower"
      }
    }
  });
  //decimal mask
   Inputmask("(.999){+|1},00", {
        positionCaretOnClick: "radixFocus",
        radixPoint: ",",
        _radixDance: true,
        numericInput: true,
        placeholder: "0",
        definitions: {
            "0": {
                validator: "[0-9\uFF11-\uFF19]"
            }
        }
   }).mask(selector);
});
```

Alternator masks

The alternator syntax is like an **OR** statement. The mask can be one of the 3 choices specified in the alternator.

To define an alternator use the |. ex: "a|9" => a or 9 "(aaa)|(999)" => aaa or 999 "(aaa|999|9AA)" => aaa or 999 or 9AA

Also make sure to read about the keepStatic option.

```
$("selector").inputmask("(99.9)|(X)", {
  definitions: {
    "X": {
      validator: "[xX]",
      casing: "upper"
    }
  }
});
```

or

```
$("selector").inputmask({
  mask: ["99.9", "X"],
  definitions: {
    "X": {
      validator: "[xX]",
      casing: "upper"
    }
  }
});
```

Preprocessing masks

You can define the mask as a function which can allow to preprocess the resulting mask. Example sorting for multiple masks or retrieving mask definitions dynamically through ajax. The preprocessing fn should return a valid mask definition.

```
$(selector).inputmask({ mask: function () { /* do stuff */ return ["[1-]AAA-999", "[1-]999-AAA"]; }});
```

JIT Masking

Just in time masking. With the jitMasking option you can enable jit masking. The mask will only be visible for the user entered characters. Default: false

Value can be true or a threshold number or false.

```
Inputmask("datetime", { jitMasking: true }).mask(selector);
```

Define custom definitions

You can define your own definitions to use in your mask.
Start by choosing a masksymbol.

validator(chrs, maskset, pos, strict, opts)

Next define your validator. The validator can be a regular expression or a function.

The return value of a validator can be true, false or a command object.

Options of the command object

- pos : position to insert
- c : character to insert
- caret : position of the caret
- remove : position(s) to remove
  - pos or [pos1, pos2]
- insert : position(s) to add :
  - { pos : position to insert, c : character to insert, fromIsValid : true/false, strict : true/false }
  - [{ pos : position to insert, c : character to insert, fromIsValid : true/false, strict : true/false }, { ...}, ...]

  fromIsValid & strict
- refreshFromBuffer :

    - true => refresh validPositions from the complete buffer
    - { start: , end: } => refresh from start to end

- rewritePosition: rewrite the maskPos within the isvalid function

definitionSymbol

When you insert or delete characters, they are only shifted when the definition type is the same. This behavior can be overridden by giving a definitionSymbol. (see example x, y, z, which can be used for ip-address masking, the validation is different, but it is allowed to shift the characters between the definitions)

```
Inputmask.extendDefinitions({
  'f': {  //masksymbol
    "validator": "[0-9\(\)\.\+/ ]"
  },
  'g': {
    "validator": function (chrs, buffer, pos, strict, opts) {
      //do some logic and return true, false, or { "pos": new position, "c": character to place }
    }
  },
  'j': { //basic year
    validator: "(19|20)\\d{2}"
  },
  'x': {
    validator: "[0-2]",
    definitionSymbol: "i" //this allows shifting values from other definitions, with the same masksymbol
or definitionSymbol
  },
  'y': {
    validator: function (chrs, buffer, pos, strict, opts) {
      var valExp2 = new RegExp("2[0-5]|[01][0-9]");
      return valExp2.test(buffer[pos - 1] + chrs);
    },
    definitionSymbol: "i"
  },
  'z': {
    validator: function (chrs, buffer, pos, strict, opts) {
      var valExp3 = new RegExp("25[0-5]|2[0-4][0-9]|[01][0-9][0-9]");
      return valExp3.test(buffer[pos - 2] + buffer[pos - 1] + chrs);
    },
    definitionSymbol: "i"
  }
});
```

placeholder

Specify a placeholder for a definition. This can also be a function.

set defaults

Defaults can be set as below.

```
Inputmask.extendDefaults({
  'autoUnmask': true
});
Inputmask.extendDefinitions({
  'A': {
    validator: "[A-Za-z\u0410-\u044F\u0401\u0451\u00C0-\u00FF\u00B5]",
    casing: "upper" //auto uppercasing
  },
  '+': {
    validator: "[0-9A-Za-z\u0410-\u044F\u0401\u0451\u00C0-\u00FF\u00B5]",
    casing: "upper"
  }
});
Inputmask.extendAliases({
  'numeric': {
    mask: "r",
    greedy: false,
    ...
  }
});
```

But if the property is defined within an alias you need to set it for the alias definition. This is also for default plugin options. If the alias definitions extends on default options, you can only override it at alias level.

```
Inputmask.extendAliases({
  'numeric': {
    autoUnmask: true,
    allowPlus: false,
    allowMinus: false
  }
});
```

However, the preferred way to alter properties for an alias is by creating a new alias which inherits from the default alias definition.

```
Inputmask.extendAliases({
  'myNum': {
    alias: "numeric",
    placeholder: '',
    allowPlus: false,
    allowMinus: false
  }
});
```

Once defined, you can call the alias by:

```
Inputmask("myNum").mask(selector);
```

All callbacks are implemented as options. This means that you can set general implementations for the callbacks by setting a default.

```
Inputmask.extendDefaults({
  onKeyValidation: function(key, result){
    if (!result){
      alert('Your input is not valid')
    }
  }
});
```

Methods:

mask(elems)

Create a mask for the input.

```
$(selector).inputmask({ mask: "99-999-99"});
```

or

```
Inputmask({ mask: "99-999-99"}).mask(document.querySelectorAll(selector));
```

or

```
Inputmask("99-999-99").mask(document.querySelectorAll(selector));
```

or

```
var im = new Inputmask("99-999-99");
im.mask(document.querySelectorAll(selector));
```

or

```
Inputmask("99-999-99").mask(selector);
```

unmaskedvalue

Get the `unmaskedvalue`

```
$(selector).inputmask('unmaskedvalue');
```

or

```
var input = document.getElementById(selector);
if (input.inputmask)
  input.inputmask.unmaskedvalue()
```

Value unmasking

Unmask a given value against the mask.

```
var unformattedDate = Inputmask.unmask("23/03/1973", { alias: "dd/mm/yyyy"}); //23031973
```

remove

Remove the `inputmask`.

```
$(selector).inputmask('remove');
```

or

```
var input = document.getElementById(selector);
if (input.inputmask)
  input.inputmask.remove()
```

or

```
Inputmask.remove(document.getElementById(selector));
```

getemptymask

return the default (empty) mask value

```
$(document).ready(function(){
  $("#test").inputmask("999-AAA");
  var initialValue = $("#test").inputmask("getemptymask");  // initialValue  => "___-___"
});
```

hasMaskedValue

Check whether the returned value is masked or not; currently only works reliably when using jquery.val fn to retrieve the value

```
$(document).ready(function(){
  function validateMaskedValue(val){}
  function validateValue(val){}

  var val = $("#test").val();
  if ($("#test").inputmask("hasMaskedValue"))
```

```
    validateMaskedValue(val);
  else
    validateValue(val);
});
```

isComplete

Verify whether the current value is complete or not.

```
$(document).ready(function(){
  if ($(selector).inputmask("isComplete")){
    //do something
  }
});
```

getmetadata

The metadata of the actual mask provided in the mask definitions can be obtained by calling getmetadata. If only a mask is provided the mask definition will be returned by the getmetadata.

```
$(selector).inputmask("getmetadata");
```

setvalue

The setvalue functionality is to set a value to the inputmask like you would do with jQuery.val, BUT it will trigger the internal event used by the inputmask always, whatever the case. This is particular usefull when cloning an inputmask with jQuery.clone. Cloning an inputmask is not a fully functional clone. On the first event (mouseenter, focus, ...) the inputmask can detect if it where cloned and can reactivate the masking. However when setting the value with jQuery.val there is none of the events triggered in that case. The setvalue functionality does this for you.

```
$(selector).inputmask("setvalue", value);

var selector = document.getElementById("selector");
selector.inputmask.setValue(value);

Inputmask.setValue(selector, value);
```

option(options, noremask)

Get or set an option on an existing inputmask. The option method is intented for adding extra options like callbacks, etc at a later time to the mask.

When extra options are set the mask is automatically reapplied, unless you pas true for the noremask argument.

Set an option

```
document.querySelector("#CellPhone").inputmask.option({
  onBeforePaste: function (pastedValue, opts) {
    return phoneNumOnPaste(pastedValue, opts);
  }
});

$("#CellPhone").inputmask("option", {
  onBeforePaste: function (pastedValue, opts) {
    return phoneNumOnPaste(pastedValue, opts);
  }
})
```

format

Instead of masking an input element it is also possible to use the inputmask for formatting given values. Think of formatting values to show in jqGrid or on other elements then inputs.

```
var formattedDate = Inputmask.format("2331973", { alias: "datetime", inputFormat: "dd/mm/yyyy"});
```

isValid

Validate a given value against the mask.

```
var isValid = Inputmask.isValid("23/03/1973", { alias: "datetime", inputFormat: "dd/mm/yyyy"});
```

Options:

placeholder

Change the mask placeholder. Default: "_"

Instead of "_", you can change the unfilled characters mask as you like, simply by adding the `placeholder` option.
For example, `placeholder: " "` will change the default autofill with empty values

```
$(document).ready(function(){
  $("#date").inputmask("99/99/9999",{ "placeholder": "*" });
});
```

or a multi-char placeholder

```
$(document).ready(function(){
  $("#date").inputmask("99/99/9999",{ "placeholder": "dd/mm/yyyy" });
});
```

optionalmarker

Definition of the symbols used to indicate an optional part in the mask.

```
optionalmarker: { start: "[", end: "]" }
```

quantifiermarker

Definition of the symbols used to indicate a quantifier in the mask.

```
quantifiermarker: { start: "{", end: "}" }
```

groupmarker

Definition of the symbols used to indicate a group in the mask.

```
groupmarker: { start: "(", end: ")" }
```

alternatormarker

Definition of the symbols used to indicate an alternator part in the mask.

```
alternatormarker: "|"
```

escapeChar

Definition of the symbols used to escape a part in the mask.

```
escapeChar: "\\"
```

See **escape special mask chars**

mask

The mask to use.

```
Inputmask({ mask: "9{*}").mask(selector);
```

regex

Use a regular expression as a mask

```
Inputmask({ regex: "[0-9]*" }).mask(selector);
```

When using shorthands be aware that you need to double escape or use String.raw with a string literal.

```
Inputmask({ regex: "\\d*" }).mask(selector);
~
Inputmask({ regex: String.raw`\d*` }).mask(selector);
```

oncomplete

Execute a function when the mask is completed

```
$(document).ready(function(){
  $("#date").inputmask("99/99/9999",{ "oncomplete": function(){ alert('inputmask complete'); } });
});
```

onincomplete

Execute a function when the mask is incomplete. Executes on blur.

```
$(document).ready(function(){
  $("#date").inputmask("99/99/9999",{ "onincomplete": function(){ alert('inputmask incomplete'); } });
});
```

oncleared

Execute a function when the mask is cleared.

```
$(document).ready(function(){
  $("#date").inputmask("99/99/9999",{ "oncleared": function(){ alert('inputmask cleared'); } });
});
```

repeat

Mask repeat function. Repeat the mask definition x-times.

```
$(document).ready(function(){
  $("#number").inputmask({ "mask": "9", "repeat": 10 });  // ~ mask "9999999999"
});
```

greedy

Default: false Toggle to allocate as much possible or the opposite. Non-greedy repeat function.

```
$(document).ready(function(){
  $("#number").inputmask({ "mask": "9", "repeat": 10, "greedy": false });  // ~ mask "9" or mask "99" or
... mask "9999999999"
});
```

With the non-greedy option set to false, you can specify * as repeat. This makes an endless repeat.

autoUnmask

Automatically unmask the value when retrieved.
Default: false.

When setting this option to true the plugin also expects the initial value from the server to be unmasked.

removeMaskOnSubmit

Remove the mask before submitting the form.
Default: false

clearMaskOnLostFocus

Remove the empty mask on blur or when not empty removes the optional trailing part Default: true

```
$(document).ready(function(){
  $("#ssn").inputmask("999-99-9999",{placeholder:" ", clearMaskOnLostFocus: true }); //default
});
```

insertMode

Toggle to insert or overwrite input.
Default: true.
This option can be altered by pressing the Insert key.

clearIncomplete

Clear the incomplete input on blur

```
$(document).ready(function(){
  $("#date").inputmask("99/99/9999",{ "clearIncomplete": true });
});
```

aliases

Definitions of aliases.

With an alias you can define a complex mask definition and call it by using an alias name. So this is mainly to simplify the use of your masks. Some aliases found in the extensions are: email, currency, decimal, integer, date, datetime, dd/mm/yyyy, etc.

First you have to create an alias definition. The alias definition can contain options for the mask, custom definitions, the mask to use etc.

When you pass in an alias, the alias is first resolved and then the other options are applied. So you can call an alias and pass another mask to be applied over the alias. This also means that you can write aliases which "inherit" from another alias.

Some examples can be found in jquery.inputmask.xxx.extensions.js

use:

```
$("#date").inputmask("datetime");
```

or

```
$("#date").inputmask({ alias: "datetime"});
```

You can also call an alias and extend it with some more options

```
$("#date").inputmask("datetime", { "clearIncomplete": true });
```

or

```
$("#date").inputmask({ alias: "datetime", "clearIncomplete": true });
```

alias

The alias to use.

```
$("#date").inputmask({ alias: "email"});
```

onKeyDown

Callback to implement autocomplete on certain keys for example

Function arguments: event, buffer, caretPos, opts
Function return:

onBeforeMask

Executes before masking the initial value to allow preprocessing of the initial value.

Function arguments: initialValue, opts
Function return: processedValue

```
$(selector).inputmask({
  alias: 'phonebe',
  onBeforeMask: function (value, opts) {
    var processedValue = value.replace(/^0/g, "");
    if (processedValue.indexOf("32") > 1 ||     processedValue.indexOf("32") == -1) {
      processedValue = "32" + processedValue;
    }

    return processedValue;
  }
});
```

onBeforePaste

This callback allows for preprocessing the pasted value before actually handling the value for masking. This can be usefull for stripping away some characters before processing.

Function arguments: pastedValue, opts
Function return: processedValue

```
$(selector).inputmask({
  mask: '9999 9999 9999 9999',
  placeholder: ' ',
  showMaskOnHover: false,
  showMaskOnFocus: false,
  onBeforePaste: function (pastedValue, opts) {
    var processedValue = pastedValue;

    //do something with it
    return processedValue;
  }
});
```

You can also disable pasting a value by returning false in the onBeforePaste call.

Default: Calls the onBeforeMask

onBeforeWrite

Executes before writing to the masked element

Use this to do some extra processing of the input. This can be usefull when implementing an alias, ex. decimal alias, autofill the digits when leaving the inputfield.

Function arguments: event, buffer, caretPos, opts
Function return: command object (see Define custom definitions)

onUnMask

Executes after unmasking to allow post-processing of the unmaskedvalue.

Function arguments: maskedValue, unmaskedValue
Function return: processedValue

```
$(document).ready(function(){
  $("#number").inputmask("decimal", { onUnMask: function(maskedValue, unmaskedValue) {
    //do something with the value
    return unmaskedValue;
  }});
});
```

showMaskOnFocus

Shows the mask when the input gets focus. (default = true)

```
$(document).ready(function(){
  $("#ssn").inputmask("999-99-9999",{ showMaskOnFocus: true }); //default
});
```

To make sure no mask is visible on focus also set the showMaskOnHover to false. Otherwise hovering with the mouse will set the mask and will stay on focus.

showMaskOnHover

Shows the mask when hovering the mouse. (default = true)

```
$(document).ready(function(){
  $("#ssn").inputmask("999-99-9999",{ showMaskOnHover: true }); //default
});
```

onKeyValidation

Callback function is executed on every keyvalidation with the key & result as parameter.

```
$(document).ready(function(){
  $("#ssn").inputmask("999-99-9999", {
    onKeyValidation: function (key, result) {
      console.log(key + " - " + result);
    }
  });
});
```

skipOptionalPartCharacter

numericInput

Numeric input direction. Keeps the caret at the end.

```
$(document).ready(function(){
  $(selector).inputmask('€ 999.999.999,99', { numericInput: true });    //123456  =>  € ___.__1.234,56
});
```

rightAlign

Align the input to the right

By setting the rightAlign you can specify to right align an inputmask. This is only applied in combination op the numericInput option or the dir-attribute. Default is true.

```
$(document).ready(function(){
  $(selector).inputmask('decimal', { rightAlign: false });  //disables the right alignment of the decimal
input
});
```

undoOnEscape

Make escape behave like undo. (ctrl-Z)
Pressing escape reverts the value to the value before focus.
Default: true

radixPoint (numerics)

Define the radixpoint (decimal separator)
Default: ""

groupSeparator (numerics)

Define the groupseparator
Default: ""

keepStatic

Default: null (~false) Use in combination with the alternator syntax Try to keep the mask static while typing. Decisions to alter the mask will be postponed if possible.

ex. $(selector).inputmask({ mask: ["+55-99-9999-9999", "+55-99-99999-9999",], keepStatic: true });

typing 1212345123 => should result in +55-12-1234-5123 type extra 4 => switch to +55-12-12345-1234

When passing multiple masks (an array of masks) keepStatic is automatically set to true unless explicitly set through the options.

positionCaretOnTab

When enabled the caret position is set after the latest valid position on TAB Default: true

tabThrough

Allows for tabbing through the different parts of the masked field.
Default: false

definitions

ignorables

isComplete

With this call-in (hook) you can override the default implementation of the isComplete function.
Args => buffer, opts Return => true|false

```
$(selector).inputmask({
  regex: "[0-9]*",
  isComplete: function(buffer, opts) {
    return new RegExp(opts.regex).test(buffer.join(''));
  }
});
```

postValidation

Hook to postValidate the result from isValid. Usefull for validating the entry as a whole. Args => buffer, pos, currentResult, opts
Return => true|false|command object

preValidation

Hook to preValidate the input. Useful for validating regardless the definition. Args => buffer, pos, char, isSelection, opts, maskset, caretPos => return true/false/command object When return true, the normal validation kicks in, otherwise it is skipped.

staticDefinitionSymbol

The staticDefinitionSymbol option is used to indicate that the static entries in the mask can match a certain definition. Especially usefull with alternators so that static element in the mask can match another alternation.

In the example below we mark the spaces as a possible match for the "i" definition. By doing so the mask can alternate to the second mask even when we typed already "12 3".

```
Inputmask("(99 99 999999)|(i{+})", {
  definitions: {
    "i": {
      validator: ".",
      definitionSymbol: "*"
    }
  },
  staticDefinitionSymbol: "*"
}).mask(selector);
```

nullable

Return nothing when the user hasn't entered anything. Default: true

noValuePatching

Disable value property patching Default: false

positionCaretOnClick

Positioning of the caret on click.

Options:

- none
- lvp (based on the last valid position (default)
- radixFocus (position caret to radixpoint on initial click)
- select (select the whole input)
- ignore (ignore the click and continue the mask)

Default: "lvp"

casing

Apply casing at the mask-level. Options: null, "upper", "lower" or "title" or callback args => elem, test, pos, validPositions return charValue

```
casing: function(elem, test, pos, validPositions) {
          do some processing || upper/lower input property in the validPositions
          return elem; //upper/lower element
}
```

Default: null

inputmode

Default: "verbatim" Specify the inputmode - already in place for when browsers start to support themhttps://html.spec.whatwg.org/#input-modalities:-the-inputmode-attribute

colorMask

Default: false Create a css styleable mask.

You need to include the inputmask.css in your page to use this option.

See the inputmask.css for more info about the used styling. You can override the Inputmask.prototype.positionColorMask`if you need some custom positioning.

```
 Inputmask.prototype.positionColorMask = function (input, template) {
                template.style.left = input.offsetLeft + "px";
                template.zIndex = input.zIndex - 1;
           }
```

disablePredictiveText

Default: false Disables predictive text on mobile devices.

What it does.

- changes the input type to password => disables predictive text

- enables the colorMask option which creates a div, which surrounds the input. So we type in the hidden password input and render the mask in the a created div.

To use the colorMask, you need to include the inputmask.css you might need to add some css-tweaks to make it all visually correct in your page.

importDataAttributes

Specify to use the data-inputmask attributes or to ignore them.

If you don't use data attributes you can disable the import by specifying importDataAttributes: false.

Default: true

shiftPositions

Alter the behavior of the char shifting on entry or deletion.

In some cases shifting the mask entries or deletion should be more restrictive.
Ex. date masks. Shifting month to day makes no sense

Default: true

true = shift on the "def" match false = shift on the "nativeDef" match

General

set a value and apply mask

this can be done with the traditional jquery.val function (all browsers) or JavaScript value property for browsers which implement lookupGetter or getOwnPropertyDescriptor

```
$(document).ready(function(){
  $("#number").val(12345);

  var number = document.getElementById("number");
  number.value = 12345;
});
```

with the autoUnmaskoption you can change the return of $.fn.val (or value property) to unmaskedvalue or the maskedvalue

```
$(document).ready(function(){
  $('#<%= tbDate.ClientID%>').inputmask({ "mask": "99/99/9999", 'autoUnmask' : true});    //  value: 
23/03/1973
  alert($('#<%= tbDate.ClientID%>').val());    // shows 23031973     (autoUnmask: true)

  var tbDate = document.getElementById("<%= tbDate.ClientID%>");
  alert(tbDate.value);    // shows 23031973     (autoUnmask: true)
});
```

escape special mask chars

If you want a mask element to appear as a static element you can escape them by \

```
$(document).ready(function(){
  $("#months").inputmask("m \\months");
});
```

auto-casing inputmask

You can define within a definition to automatically apply some casing on the entry in an input by giving the casing. Casing can be null, "upper", "lower" or "title".

```
Inputmask.extendDefinitions({
  'A': {
    validator: "[A-Za-z]",
    casing: "upper" //auto uppercasing
  },
  '+': {
    validator: "[A-Za-z\u0410-\u044F\u0401\u04510-9]",
    casing: "upper"
  }
});
```

Include jquery.inputmask.extensions.js for using the A and # definitions.

```
$(document).ready(function(){
  $("#test").inputmask("999-AAA");    //   => 123abc ===> 123-ABC
});
```

Supported markup options

RTL attribute

```
<input id="test" dir="rtl" />
```

readonly attribute

```
<input id="test" readonly="readonly" />
```

disabled attribute

```
<input id="test" disabled="disabled" />
```

maxlength attribute

```
<input id="test" maxlength="4" />
```

data-inputmask attribute

You can also apply an inputmask by using the data-inputmask attribute. In the attribute you specify the options wanted for the inputmask. This gets parsed with $.parseJSON (for the moment), so be sure to use a well-formed json-string without the {}.

```
<input data-inputmask="'alias': 'datetime'" />
<input data-inputmask="'mask': '9', 'repeat': 10, 'greedy' : false" />

$(document).ready(function(){
  $(":input").inputmask();
});
```

data-inputmask-<option> attribute

All options can also be passed through data-attributes.

```
<input data-inputmask-mask="9" data-inputmask-repeat="10" data-inputmask-greedy="false" />
```

```
$(document).ready(function(){
  $(":input").inputmask();
});
```

jQuery.clone

When cloning a inputmask, the inputmask reactivates on the first event (mouseenter, focus, ...) that happens to the input. If you want to set a value on the cloned inputmask and you want to directly reactivate the masking you have to use $(input).inputmask("setvalue", value)

Be sure to pass true in the jQuery.clone fn to clone with data and events and use jQuery as dependencyLib (https://api.jquery.com/clone/)

Extensions

datetime extensions

Date and Time masks.

Aliases

- datetime

Options

inputFormat

Format used to input the date

ex:

- dd/mm/yyyy
- mm/dd/yyyy
- dd.mm.yyyy HH:MM:ss

###Supported symbols

- d
  Day of the month as digits; no leading zero for single-digit days.
- dd
  Day of the month as digits; leading zero for single-digit days.
- ddd
  Day of the week as a three-letter abbreviation.
- dddd
  Day of the week as its full name.
- m
  Month as digits; no leading zero for single-digit months.
- mm
  Month as digits; leading zero for single-digit months.
- mmm
  Month as a three-letter abbreviation.
- mmmm
  Month as its full name. -yy
  Year as last two digits; leading zero for years less than 10.

- yyyy
  Year as 4 digits.
- h
  Hours; no leading zero for single-digit hours (12-hour clock).
- hh
  Hours; leading zero for single-digit hours (12-hour clock).
- hhh
  Hours; no limit -H
  Hours; no leading zero for single-digit hours (24-hour clock).
- HH
  Hours; leading zero for single-digit hours (24-hour clock).
- HHH
  Hours; no limit
- M
  Minutes; no leading zero for single-digit minutes. Uppercase M unlike CF timeFormat's m to avoid conflict with months.
- MM
  Minutes; leading zero for single-digit minutes. Uppercase MM unlike CF timeFormat's mm to avoid conflict with months.
- s
  Seconds; no leading zero for single-digit seconds.
- ss
  Seconds; leading zero for single-digit seconds.
- l
  Milliseconds. 3 digits.
- L
  Milliseconds. 2 digits.
- t
  Lowercase, single-character time marker string: a or p.
- tt
  Two-character time marker string: am or pm.
- T
  Single-character time marker string: A or P.
- TT
  Two-character time marker string: AM or PM.
- Z
  US timezone abbreviation, e.g. EST or MDT. With non-US timezones or in the Opera browser, the GMT/UTC offset is returned, e.g. GMT-0500
- o
  GMT/UTC timezone offset, e.g. -0500 or +0230.
- S
  The date's ordinal suffix (st, nd, rd, or th). Works well with d.

Optional parts

To mark a part of the inputFormat as optional, use the [] as you would for other masks.

Ex. inputFormat: "dd/mm/yyyy [HH]"

displayFormat

Visual format when the input looses focus

outputFormat

Unmasking format

min

Minimum value. This needs to be in the same format as the inputfornat

max

Maximum value. This needs to be in the same format as the inputfornat,

numeric extensions

Aliases

- numeric
- currency
- decimal
- integer
- percentage

Options

digits

Number of fractionalDigits Default: "*"

The value can be a number, *, or a quantifier syntax like 2,4 When the quantifier syntax is used, the digitsOptional option is ignored

digitsOptional

Specify wheter the digits are optional. Default: true

enforceDigitsOnBlur

Enforces the decimal part when leaving the input field.

allowMinus

Allow to enter -. Default: true

negationSymbol

Define your negationSymbol. Default: { front: "-", //"(" back: "" //")" }

prefix

Define a prefix. Default: ""

suffix

Define a suffix. Default: ""

min

Minimum value Default: undefined

max

Maximum value Default: undefined

step

Define the step the ctrl-up & ctrl-down must take. Default: 1

unmaskAsNumber

Make unmasking returning a number instead of a string. Default: false

Be warned that using the unmaskAsNumber option together with jQuery.serialize will fail as serialize expects a string. (See issue #1288)

inputType

Indicates whether the value passed for initialization is text or a number

Default: "text"

Setting initial values

When initializing the mask with a value, you need to take some rules into account. Depending of the option inputType the value will be interpreted as text or as a number.

When inputType is text, the symbol of the radixPoint must be correct. When using number the . (dot) is used as radixpoint.

Setting a number will always work when using vanilla javascript setters.

Example with komma (,) as radixpoint

```
/html
<input name="npt" value="123456,789"/>

//js
Inputmask("decimal", {
    radixPoint: ',',
    inputtype: "text"
}).mask("input");

$("input").val("123456,789");
$("input").val(123456.789); //this doesn't work because jQuery converts the number to a string
before passing it along to the Inputmask.

document.getElementsByName("npt")[0].value = "123456,789";
document.getElementsByName("npt")[0].value = 123456.789; //type number
```

other extensions

Definitions

- A : alphabetical uppercasing
- & : alfanumeric uppercasing
- # : hexadecimal

Aliases

URL

An URL mask for entering valid FTP, HTTP or HTTPS addresses.

```
Inputmask("url").mask(selector);
```

IP address

An IP address alias for entering valid IP addresses.

```
Inputmask("ip").mask(selector);
```

Email

An email mask for entering valid email addresses.

```
Inputmask("email").mask(selector);
```

MAC

An MAC mask for entering valid MAC addresses.

```
Inputmask("mac").mask(selector);
```

VIN (Vehicle identification number)

An VIN mask for entering valid VIN codes.

```
Inputmask("vin").mask(selector);
```

You can find/modify/extend these aliases in the inputmask.extensions.js