

Getting started

Components

- M4Q
- About
- Population
- Constructor
- Ajax
- Animation
- Loops
- Visibility
- Effects
- Subtree functions
- Attributes
- Html, text and value
- Css and classes
- Position and size
- Manipulation
- DataSet
- Events
- Utils

- Base
- Containers
- Grid system
- Typography
- Tables
- Forms
- Buttons
- Images
- Figures
- Lists

- Form components
- Checkbox
- File input
- Input
- Input material
- Keypad
- Rating
- Radio
- Select
- Slider
- Double Slider
- Spinner
- Switch
- Tag input
- Textarea

- Menus
- App bar
- Bottom navigation
- Bottom sheet
- Menu
- Ribbon Menu
- Side bar
- Side navigation

- Controls
- Accordion
- Badge
- Carousel
- Cards
- Cube
- Counter
- Charms
- Chat
- Donut
- Image compare
- Image magnifier
- Gravatar
- List
- ListView
- Master
- NavView
- Panels
- Progress & Activity
- Streamer
- Stepper
- Splitter
- Tabs
- Tab material
- Table
- Tiles
- TreeView
- Wizard
- Information
- Dialogs
- Info box
- Hints
- Notify system
- Popovers
- Toasts
- Windows
- Date & time
- Calendar
- Calendar picker
- Date picker
- Time picker
- Countdown
- Formatting date

- Media
- Video player
- Audio player

- Tools
- Collapse
- Color module
- Draggable
- Dropdown

Forms

Examples and usage guidelines for form control styles, layout options, and custom components for creating a wide variety of forms.

Overview

Metro's form controls expand on our Rebooted form styles with classes. Use these classes to opt into their customized displays for a more consistent rendering across browsers and devices.

Be sure to use an appropriate `type` attribute on all inputs (e.g., `email` for email address or `number` for numerical information) to take advantage of newer input controls like email verification, number selection, and more.

Email address

Enter email

We'll never share your email with anyone else.

Password

Enter email

☐ Remember me

Submit data

Cancel

```
<form>
  <div class="form-group">
    <label>Email address</label>
    <input type="email" placeholder="Enter email"/>
    <small class="text-muted">We'll never share your email with anyone else.</small>
  </div>
  <div class="form-group">
    <label>Password</label>
    <input type="password" placeholder="Enter email"/>
  </div>
  <div class="form-group">
    <input type="checkbox" data-role="checkbox" data-caption="Remember me">
  </div>
  <div class="form-group">
    <button class="button success">Submit data</button>
    <input type="button" class="button" value="Cancel">
  </div>
</form>
```

Form controls

Metro 4 styling all forms controls with direct styles or special classes. Now supports: all types of `<input>`, `<select>`, `<textarea>`, `<checkbox>` and `<radio>`. Included are styles for general appearance, focus state, sizing, and more.

Inputs

You can use default `input` or `extended input`.

Default input

```
<form>
  <input type="text" class="mt-1">
</form>
```

Extended input with data-role="input"

Input

Password

Input with prepend

Email

Input with append

Email

Input with append & prepend

\$

0.00

Custom button

```
<input type="text" data-role="input">

<input type="password" data-role="input">

<input type="text" data-role="input"
  data-prepend="<span class='mif-user'></span>">

<input type="text" data-role="input" data-prepend="Email">
<input type="text" data-role="input" data-append="Email">

<input type="text" data-role="input"
  class="mb-1"
  data-prepend="<span class='mif-dollar2'></span>"
  data-append="<span>0.00</span>" title="">

<script>
```

Table of contents

- Forms
- Overview
- Form controls
- Form controls
- Input
- Tag input
- Spinner
- Search
- Select
- Textarea
- Checkbox
- Radio
- Switch
- File browser
- Buttons
- Layout
- Form groups
- Grid layout
- Horizontal form
- Inline form
- Accent colors
- Disabled forms
- Validation
- How it works
- Browser defaults
- Use the power, Luke!
- Rtl support

```
var customButtons = [
  {
    html: "<span class='mif-user'></span>",
    cls: "alert",
    onclick: "alert('ku from custom button')"
  }
]
</script>
<input type="text"
  data-role="input"
  data-clear-button="false"
  data-custom-buttons="customButtons">
```

Additional functionality for input is implemented through the plugin `input`. To activate plugin and additional options add the attribute `data-role="input"` to element.

Tag input

New in 4.2.18

To create tag input, add attribute `data-role="taginput"` to input field. Enter tag name and press `Enter` or `,`

```
<form>
  <input type="text" data-role="taginput">
</form>
```

Init values

css x javascript x html x metro 4 x

Specific tags colors

Random tag colors

Events

```
<p>Init values</p>
<input type="text" data-role="taginput" class="mt-4" value="css, javascript, html, metro 4">

<p>Specific tags colors</p>
<input type="text" data-role="taginput" class="mt-4" data-cls-tag="bg-cyan fg-white" data-cls-tag-remover="bg-darkCyan fg-white">

<p>Random tag colors</p>
<input type="text" data-role="taginput" class="mt-4" data-random-color="true">

<p>Events</p>
<input type="text" data-role="taginput" class="mt-4" data-on-tag="$('#output').text(this.value)">
<div class="debug p-10 mt-4 text-center" id="output"></div>
```

Spinner

New in 4.2.20

To create `spinner`, add attribute `data-role="spinner"` to input field.

```
<input type="text" data-role="spinner">
```

Min - 0, max - 5

Step - 0.52, fixed - 2

Buttons left

Buttons right

Additional classes

```
<p>Min - 0, max - 5</p>
<input type="text" data-role="spinner" data-min-value="0" data-max-value="5">

<p>Step - 0.52, fixed - 2</p>
<input type="text" data-role="spinner" data-step=".52" data-fixed="2">

<p>Buttons left</p>
<input type="text" data-role="spinner" data-buttons-position="left">

<p>Buttons right</p>
<input type="text" data-role="spinner" data-buttons-position="right">

<p>Additional classes</p>
<input type="text" data-role="spinner"
  data-cls-spinner-value="text-bold bg-cyan fg-white"
  data-cls-spinner-button="info"
  data-plus-icon="<span class='mif-plus fg-white'></span>"
  data-minus-icon="<span class='mif-minus fg-white'></span>">
```

+ -

+ -

+ -

+ -

+ -

+ -

You can set or get `spinner` value with method `val()` . Also you can set or get value with input attribute `value` . To set to default value, you can use attribute `data-default-value` and method `toDefault()` .

Search

You can define `input` as `search-input` . To create search input, enable `search-button` with attribute `data-search-button="true"` .

```
<form>
  <input type="text" data-role="input" data-search-button="true">
</form>
```

The `search button` will be of the `submit` type if you specify an attribute `data-search-button-click="submit"` . If you set this attribute to other value, you can specify custom event for click event on search button with attribute `data-on-search-button-click="..."` .

Select

Metro 4 uses a special wrapper to display the drop-down select. When you use Metro 4 select wrapper, you can styling every option with css classes. Use attribute `data-role="select"` to activate wrapper.

Single

Two

▼eVPS-1

```
<select data-role="select">
  <option class="fg-cyan">One</option>
  <option selected class="text-bold fg-red">Two</option>
  <option class="fg-green">Three</option>
</select>

<select data-role="select">
  <optgroup label="Virtual hosting">
    <option value="mini">Mini</option>
    <option value="site">Site</option>
    <option value="portal">Portal</option>
  </optgroup>
  <optgroup label="Virtual servers">
    <option value="evps0">eVPS-TEST (30 дней)</option>
    <option value="evps1" selected="selected">eVPS-1</option>
    <option value="evps2">eVPS-2</option>
    <option value="evps4">eVPS-4</option>
    <option value="evps8">eVPS-8</option>
    <option value="evps16">eVPS-16</option>
    <option value="evps32">eVPS-32</option>
    <option value="evps64">eVPS-64</option>
    <option value="evps128">eVPS-128</option>
  </optgroup>
</select>
```

Multiple

New in 4.2.18

Metro 4 also support select with `multiple` option. Each selected option in multiple displayed as tag.

eVPS-1 x


▼

```
<select data-role="select" title="" multiple>
  <optgroup label="Physical servers">
    <option value="dedicated_corei3_hp">Core i3 (hp)</option>
    <option value="dedicated_pentium_hp">Pentium (hp)</option>
    <option value="dedicated_smart_corei3_hp">Smart Core i3 (hp)</option>
    <option value="dedicated_smart_pentium_hp">Smart Pentium (hp)</option>
    <option value="dedicated_smart_xeon_hp">Smart Xeon (hp)</option>
    <option value="dedicated_xeon_hp">Xeon (hp)</option>
    <option value="dedicated_xeon">Xeon</option>
    <option value="dedicated_xeon_sata">Xeon SATA</option>
    <option value="dedicated_dual_xeon">Dual Xeon</option>
    <option value="dedicated_dual_xeon_sata">Dual Xeon SATA</option>
    <option value="dedicated_smart_xeon">Smart Xeon</option>
    <option value="dedicated_smart_xeon_sata">Smart Xeon SATA</option>
    <option value="dedicated_smart_dual_xeon">Smart Dual Xeon</option>
    <option value="dedicated_smart_dual_xeon_sata">Smart Dual Xeon SATA</option>
    <option value="colo_new_2u">Colocation 2U</option>
    <option value="colo_new_4u">Colocation 4U</option>
  </optgroup>
  <optgroup label="Virtual hosting">
    <option value="mini">Mini</option>
    <option value="site">Site</option>
    <option value="portal">Portal</option>
  </optgroup>
  <optgroup label="Virtual servers">
    <option value="evps0">eVPS-TEST (30 дней)</option>
    <option value="evps1" selected="selected">eVPS-1</option>
    <option value="evps2">eVPS-2</option>
    <option value="evps4">eVPS-4</option>
    <option value="evps8">eVPS-8</option>
    <option value="evps16">eVPS-16</option>
    <option value="evps32">eVPS-32</option>
    <option value="evps64">eVPS-64</option>
    <option value="evps128">eVPS-128</option>
  </optgroup>
</select>
```

Option templates

New in 4.2.18

You can use attribute `data-template` for option for define html option display. Example: you need to add `icon` to option. `$1` in template, used for replace by option text.

 eVPS-1

▼

```
<select data-role="select" title="">
  <optgroup label="Physical servers">
    <option value="11" data-template="<span class='mif-server icon'></span> $1">Core i3 (hp)</option>
    <option value="12" data-template="<span class='mif-server icon'></span> $1">Pentium (hp)</option>
```

```
<option value="13" data-template="<span class='mif-server icon'></span> $1">Smart Core i3 (hp)</option>
<option value="14" data-template="<span class='mif-server icon'></span> $1">Smart Pentium (hp)</option>
</optgroup>
<optgroup label="Virtual servers">
  <option value="21" data-template="<span class='mif-insert-template icon'></span> $1">eVFS-TEST (30 дней)</option>
  <option value="22" selected="selected" data-template="<span class='mif-insert-template icon'></span> $1">eVFS-1</option>
  <option value="23" data-template="<span class='mif-insert-template icon'></span> $1">eVFS-2</option>
  <option value="24" data-template="<span class='mif-insert-template icon'></span> $1">eVFS-4</option>
</optgroup>
</select>
```

Additional functionality for select is implemented through the plugin `select`. To activate plugin and additional options add the attribute `data-role="select"` to element.

In version `4.2.0`, component receive new feature: `filtering`.

Select in runtime

To set or get `select` value at runtime, use method `val()`. Calling a method without a parameter will return the current `select` value. Calling a method with a parameter will set value for `select`.

```
<select data-role="select">
  <option value="1">Value 1</option>
  <option value="2">Value 2</option>
  <option value="3">Value 3</option>
  <option value="4">Value 4</option>
</select>
```

```
function selectChangeValue(){
  var new_value = Metro.utils.random(1, 4);
  var select = $('#select').data('select');

  console.log("Current value: " + select.val());
  console.log("New value: " + new_value);
  select.val(new_value);
  console.log("Result value: " + select.val());
}
```

To change select options at runtime, use method `data()`. You can use `PLAIN HTML` or `JSON OBJECT` to update options. When you execute method `data()`, options will be replaced in select. For more info see source if [this example](#).

HTML example

```
<option value="mini">Mini</option>
<option value="site">Site</option>
<option value="portal">Portal</option>
<option value="portal">Corporate</option>
<optgroup label="Virtual servers">
  <option value="evps1">eVPS-1</option>
  <option value="evps2">eVPS-2</option>
  <option value="evps4">eVPS-4</option>
  <option value="evps8">eVPS-8</option>
</optgroup>
```

JSON example

```
{
  "1": "Item 1",
  "2": "Item 2",
  "3": "Item 3",
  "Item group": {
    "1_1": "Item 1_1",
    "2_2": "Item 2_2",
    "3_3": "Item 3_3"
  }
}
```

```
<select id="select1" data-role="select">...</select>
...
$.get(_url_, function(response){
  var select = $('#select1').data('select');
  select.data(response);
});
```

Textarea

Metro 4 uses a special wrapper to display the textarea. When you use Metro 4 textarea wrapper, you can enabling `autosize` feature. Use attribute `data-role="textarea"` to activate wrapper.

Default

Autosize

 Autosize

Disabled

```
<textarea data-role="textarea"></textarea>
<textarea data-role="textarea" data-auto-size="true" data-max-height="200"></textarea>
<textarea data-role="textarea" disabled></textarea>
```

Additional functionality for textarea is implemented through the plugin `textarea`. To activate plugin and additional options add the attribute `data-role="textarea"` to element.

Checkbox

Default checkbox is not styled, but you can add attribute `data-role="checkbox"` to element to styling.

Default checks

☐ ☒ ☐ ☐

Metro 4 checkboxes

Style 1

☐ Unchecked ☒ Checked ☐ Indeterminate ☐ Disabled ☒ Disabled checked

Style 2 New in 4.2.18

☐ Unchecked ☒ Checked ☐ Indeterminate ☐ Disabled ☒ Disabled checked

```
<input type="checkbox" data-role="checkbox" data-caption="Check">
<input type="checkbox" data-role="checkbox" data-caption="Check" checked>
<input type="checkbox" data-role="checkbox" data-caption="Check" data-indeterminate="true">
<input type="checkbox" data-role="checkbox" data-caption="Check" disabled>
<input type="checkbox" data-role="checkbox" data-caption="Check" disabled checked>

<input type="checkbox" data-role="checkbox" data-caption="Check" data-style="2">
```

Additional functionality for input is implemented through the plugin `checkbox`. To activate plugin and additional options add the attribute `data-role="checkbox"` to element.

Indeterminate state

In Metro 4, the checkbox can have an indeterminate state. To set this state add or remove attribute `data-indeterminate="true"` for/from input.

Indeterminate

Set state

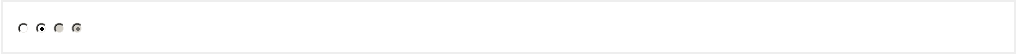
Remove state

```
<input type="checkbox"
  data-role="checkbox"
  data-caption="Indeterminate"
  indeterminate
  id="indeterminate-checkbox">
<div class="mt-2">
  <button class="button"
    onclick="$('#indeterminate-checkbox').toggleAttr('data-indeterminate', 'true')">
  >Set state</button>
  <button class="button"
    onclick="$('#indeterminate-checkbox').toggleAttr('data-indeterminate', 'false')">
  >Remove state</button>
</div>
```

Radio

Default radio is not styled, but you can add attribute `data-role="radio"` to element to styling.

Default radio



Styled radio

Style 1

☐ Check

☒ Check

☐ Check

☐ Check

Style 2

New in 4.2.18

☐ Check

☒ Check

☐ Check

☐ Check

```
<input type="radio" data-role="radio" data-caption="Check" name="r1">
<input type="radio" data-role="radio" data-caption="Check" name="r1" checked>
<input type="radio" data-role="radio" data-caption="Check" name="r2" disabled>
<input type="radio" data-role="radio" data-caption="Check" name="r2" checked disabled>

<input type="radio" data-role="radio" data-caption="Check" name="r1" data-style="2">
```

Additional functionality for input is implemented through the plugin `radio`. To activate plugin and additional options add the attribute `data-role="radio"` to element.

Switch

Metro 4 support additional styling for checkbox. Use attribute `data-role="switch"` to create checkbox styling as `switch`.

☒ Check

☐ Check

☐ Check

☐ Check

```
<input type="checkbox" data-role="switch" data-caption="Check">
<input type="checkbox" data-role="switch" data-caption="Check" checked>
<input type="checkbox" data-role="switch" data-caption="Check" disabled>
<input type="checkbox" data-role="switch" data-caption="Check" disabled checked>
```

Additional functionality for input is implemented through the plugin `switch`. To activate plugin and additional options add the attribute `data-role="switch"` to element.

File browser

Select your photo:

```
<input type="file" data-role="file">
<input type="file" data-role="file" data-button-title="Choose file">
<input type="file" data-role="file" data-button-title="<span class='mif-folder'></span>">
```

Choose files



Additional functionality for input is implemented through the plugin `file`. To activate plugin and additional options add the attribute `data-role="file"` to element.

Form buttons

Button

Reset

Submit

```
<input type="button" class="button" value="Button">
<input type="reset" class="button" value="Reset">
<input type="submit" class="button" value="Submit">
```

Layout

Since Metro 4 applies `display: block` and `width: 100%` to almost all our form controls, forms will by default stack vertically. Additional classes can be used to vary this layout on a per-form basis.

Form groups

The `.form-group` class is the easiest way to add some structure to forms. Its only purpose is to provide margin-bottom around a label and control pairing. As a bonus, since it's a class you can use it with `<fieldset>`, `<div>`, or nearly any other element.

Example label

Example label

```
<form>
  <div class="form-group">
    <label>Example label</label>
    <input type="text">
  </div>
  <div class="form-group">
    <label>Example label</label>
    <input type="text">
  </div>
</form>
```

Grid layout

More complex forms can be built using our grid classes. Use these for form layouts that require multiple columns, varied widths, and additional alignment options.

First name

Last name

```
<form>
  <div class="row">
    <div class="cell-md-6">
      <input type="text" placeholder="First name">
    </div>
    <div class="cell-md-6">
      <input type="text" placeholder="Last name">
    </div>
  </div>
</form>
```

Horizontal form

Create horizontal forms with the grid by adding the `.row` class to form groups and using the `.cell-*-*` classes to specify the width of your labels and controls.

Email

Password

Radios

☒ Option one ☐ Option two ☐ Option three is disabled

Checkbox

☒ Check me out

Sign in

```
<form>
  <div class="row mb-2">
    <label class="cell-sm-2">Email</label>
    <div class="cell-sm-10">
      <input type="email">
    </div>
  </div>
  <div class="row mb-2">
    <label class="cell-sm-2">Password</label>
    <div class="cell-sm-10">
      <input type="password">
    </div>
  </div>
  <div class="row mb-2">
    <label class="cell-sm-2">Radios</label>
    <div class="cell-sm-10">
      <input name="horizontal_form_r1" type="radio" data-role="radio" data-caption="Option one" checked>
      <input name="horizontal_form_r1" type="radio" data-role="radio" data-caption="Option two">
      <input name="horizontal_form_r1" type="radio" data-role="radio" data-caption="Option three is disabled" disabled>
    </div>
  </div>
  <div class="row mb-2">
    <label class="cell-sm-2">Checkbox</label>
    <div class="cell-sm-10">
      <input name="horizontal_form_c1" type="checkbox" data-role="checkbox" data-caption="Check me out" checked>
    </div>
  </div>
  <div class="row">
    <div class="cell">
      <button type="submit" class="button primary">Sign in</button>
    </div>
  </div>
</form>
```

Inline form

Use the `.inline-form` class to display a series of labels, form controls, and buttons on a single horizontal row. Form controls within inline forms vary slightly from their default states.

Your Name

Username

☐ Remember me

Submit

```
<form class="inline-form">
  <input type="text" placeholder="Your Name">
  <input type="text" placeholder="Username">
  <input type="checkbox" data-role="checkbox" data-caption="Remember me">
  <button class="button success">Submit</button>
</form>
```

Accent colors

You can add accent colors to form elements such as `<input>`, `<select>` and `<textarea>`. You can use next classes: `.primary`, `.secondary`, `.success`, `.alert`, `.warning`, `.yellow`, `.info`, `.dark`.

Three

```
<input type="text" class="alert">
<input type="password" class="success" data-role="input">
<textarea data-role="textarea" class="primary"></textarea>
<select class="warning" data-role="select">
  <option class="fg-cyan">One</option>
  <option class="text-bold fg-red">Two</option>
  <option selected class="fg-green">Three</option>
</select>
```

Disabled forms

To disable form elements add the `disabled` boolean attribute to element.

One

```
<form>
  <input type="text" data-role="input" disabled class="mb-1">
  <input type="file" data-role="file" disabled class="mb-1">
  <textarea data-role="textarea" disabled class="mb-1"></textarea>
  <select data-role="select" disabled>
    <option>One</option>
    <option>Two</option>
    <option>Three</option>
  </select>
</form>
```

To disable elements at runtime:

Disable input

☐

Enable input

```
<input type="text" data-role="input" id="disable_input_runtime">
<input type="checkbox" data-role="switch" checked onclick="toggleInputState(this)">
<script>
  function toggleInputState(e1) {
    var i = $('#disable_enable_input_runtime');
    $(e1).is(':checked') ? i.attr('disabled', false) : i.attr('disabled', true);
  }
</script>
```

Validation form

If you use default form elements (without defining `data-role`), you can provide valuable, actionable feedback to your users with HTML5 form validation—available in all our supported browsers. Choose from the browser default validation feedback, or implement custom messages with our built-in classes and starter JavaScript.

We **highly recommended** [Metro 4 Validator](#) plugin or other third-party plugins for validating form.

How it works

Here's how default form validation works with Metro 4:

- HTML form validation is applied via CSS's two pseudo-classes, `:invalid` and `:valid`. It applies to `<input>`, `<select>`, and `<textarea>` elements.
- Add `.custom-validation` class to form
- Add `require` attribute to element
- To invalid feedback add text element after input (select, textarea) with class `.invalid-feedback`
- Add `novalidate` attribute to form

First name

Billy

Last name

Gates

City

City

Please provide a valid city.

State

State

Please provide a valid state.

Zip

Zip

Please provide a valid zip.

Submit form

Browser defaults

Not interested in custom validation feedback messages or writing JavaScript to change form behaviors? All good, you can use the browser defaults. Try submitting the form below. Depending on your browser and OS, you'll see a slightly different style of feedback.

While these feedback styles cannot be styled with CSS, you can still customize the feedback text through JavaScript.

First name

Billy

Last name

Gates

City

State

Zip

City	State	Zip
<input type="button" value="Submit form"/>		

Use the power, Luke!

To better support form validation, use the plugin [Metro 4 Validator](#). This plugin provides more functionality to validating form with special validation funcs include custom functions and patterns.

Label for input

Rtl support

Metro supports rtl input method. To set Metro's rtl you can add `metro-rtl.css` to page head and add attribute `dir` with `rtl` value to element.

Inputs

```
<input type="text" dir="rtl">  
<input type="text" dir="rtl" data-role="input" data-clear-button="true">
```

File browser

```
<input type="file" dir="rtl" data-role="file">
```

Choose file(s)

Textarea

```
<textarea dir="rtl"></textarea>  
<textarea data-role="textarea" dir="rtl"></textarea>
```

Select

```
<select dir="rtl">  
<option value="Two">Two</option>  
<option value="Two">Two</option>  
</select>
```

