(This document is public)

New List
Public Documentation

Visibility: PUBLIC
Status: Draft
Authors: bdiamond
Last updated: 2021-08-09

Objective

This document is intended to serve as a draft for the public List Evolution README ("Evolution" is used to describe the new implementation of List). The focus will be in documenting the List Evolution DOM and CSS structure.

README First

1. Note that the old List implementation is still available, but uses a "mdc-deprecated-list-" class name prefix (as well as a "deprecated-" prefix for Sass mixins).
2. The deprecated implementation is not described by this document, only the new (Evolution) implementation.
3. Components that use List (e.g., Menu, Select, Autocomplete) have not been updated to use List Evolution yet. Please continue using the deprecated ("mdc-deprecated-list-") implementation when working with these components.

Imminent Changes

Note that several of the tokens described below are subject to change. These tokens are correct as of the "last updated" date (above). However, certain changes are expected in the coming weeks:

- Evolution will become the default, so extra HTML data attributes will no longer be needed.
    - E.g., data-evolution=”true” will not be needed to opt-into evolution. Instead, data-deprecated=”true” will be needed to opt-into old behavior.
    - If possible, better to avoid support for deprecated code if unnecessary.

DOM structure

Each section describes the tokens (classes, attributes, and other properties) supported by a given element in the DOM. Tokens apply to the element associated with the containing section; subsections correspond to nested elements.

Paths

Each section includes a “path”, which summarizes the corresponding element’s position within the list DOM. These can be interpreted as follows:

- Carets are used to indicate parent/child relationships (`parent > child`).
- The underlined item corresponds to the element being documented (e.g, `ul > li`).
- Brackets are used to indicate ordering; siblings must be arranged in ascending order (there may be gaps if certain blocks aren’t included).
- Parens specify clarifying labels (e.g., `ul > span(start) > span`).
- Bars indicate alternatives when a block may appear in multiple parents (e.g., `ul > span(start) | span(end) > span`); only affects one entry in the path.

As an example, consider the path for primary text:

`ul > li > span(content) > span[1]`

This path indicates that primary text is contained within the content span of a list item in an unordered list. The number in brackets (e.g., `[1]`) indicates the primary text’s *relative* position in its parent. Elements with lower numbers (e.g., overline text at 0) rank before the primary text and elements with higher numbers (e.g., secondary text at 2) rank after. Note that these numbers are only considered when the corresponding element is present (e.g., most list items don’t have overline text and therefore the primary text will always be the first child).

Aspects

Aspects are used to express dependencies between tokens. These appear in brackets (e.g., “*[start]*”) in the type column. Usage notes reference these to characterize how a particular group of tokens interact (because they all affect the same aspect, or property, of the element).

For example, "*Mutually exclusive with [control], [start] aspects*" indicates that only one token tagged with the "[control]" **or** "[start]" aspects can be added to an element at a time. This specific constraint is useful because items can only contain a single leading control or media type (e.g, an icon).

Lists

List Element

Path: `ul`

Though discouraged, lists may contain items with different layouts (e.g., leading and trailing media, control position, number of lines, etc.).

Accessibility roles must never be mixed (e.g., every item in a single selection list should have a radio button). A **listbox** should only contain **option** items, a **menu** should only contain **menuitem** items, and a basic list should only contain items without a role.

Token	Type	Usage
`mdc-list`	Class	All evolution lists must have this class.
`data-evolution="true"`	Attribute	Opt into evolution (will be removed after dogfood; evolution will be the default)
`role`	Attribute	Requirements: ● Non-interactive: **do not specify** ● Menu list: **menu** ● Option list: **listbox** ● Single-selection list: **listbox** ● Multiple-selection list: **listbox** (see below for other requirements)
`aria-multiselectable="true"`	Attribute	Only specify for multiple-selection lists.
`aria-label`	Attribute	Encouraged for a11y.
`tabindex`	Attribute	The list container should delegate focus to items. Focus is assigned in this order: (1) the last selected item, (2) the last focused item, (3) the first selectable item.

Items

Item Element

Path: `ul > li`

Lists must include a minimum of one item.

Token	Type	Usage
`mdc-list`	Class	All evolution items must have this class.
`mdc-list-item--disabled`	Class	Required by disabled **interactive** items, **with or without a control**.
`mdc-list-item--selected`	Class	Required by selected **interactive** items **without a control**. Items with controls delegate selection state to the control. We currently do not
`mdc-list-item--activated`	Class	Alternate styling for selected **interactive** items **without a control**. Has different semantics than selected: ● Selection represents a choice that might change frequently (e.g., option in a list) ● Activation represents a status with more permanence (e.g., current page in a navigation list).
`mdc-list-item--non-interactive`	Class	Should be added to non-interactive lists to apply correct styling. Can only be used with the **basic** list type (not menu, option, single, or multiple selection lists).
`mdc-list-item--with-one-line`	Class *[lines]*	Required for **one-line** items. *Mutually exclusive with [lines]; must choose one.*
`mdc-list-item--with-two-li`	Class	Required for **two-line** items.

`nes`	*[lines]*	*Mutually exclusive with [lines]; must choose one.*
`mdc-list-item--with-three-lines`	Class *[lines]*	Required for **three-line** items. *Mutually exclusive with [lines]; must choose one.*
`mdc-list-item--with-leading-checkbox`	Class *[start] [control]*	Add to allow room for a **leading** checkbox. Can only be used in a **multiple selection** list. *Mutually exclusive with [start], [control].*
`mdc-list-item--with-leading-radio`	Class *[start] [control]*	Add to allow room for a **leading radio button**. Can only be used in a **single selection** list. *Mutually exclusive with [start], [control].*
`mdc-list-item--with-leading-switch`	Class *[start] [control]*	Add to allow room for a **leading** switch. **Not yet supported.** Can only be used in a **single selection** list. *Mutually exclusive with [start], [control].*
`mdc-list-item--with-leading-icon`	Class *[start]*	Add to allow room for a **leading** icon. *Mutually exclusive with [start].*
`mdc-list-item--with-leading-image`	Class *[start]*	Add to allow room for a **leading** image. *Mutually exclusive with [start].*
`mdc-list-item--with-leading-thumbnail`	Class *[start]*	Add to allow room for a **leading** thumbnail. *Mutually exclusive with [start].*
`mdc-list-item--with-leading-video`	Class *[start]*	Add to allow room for a **leading** video. *Mutually exclusive with [start].*
`mdc-list-item--with-leading-avatar`	Class *[start]*	Add to allow room for a **leading** avatar. *Mutually exclusive with [start].*
`mdc-list-item--with-traili`	Class	Add to allow room for **trailing** meta text.

`ng-meta`	*[end]*	*Mutually exclusive with [end].*
`mdc-list-item--with-trailing-icon`	Class *[end]*	Add to allow room for **trailing** icon. *Mutually exclusive with [end].*
`mdc-list-item--with-trailing-checkbox`	Class *[end] [control]*	Add to allow room for a **trailing** checkbox. Can only be used in a **multiple selection** list. *Mutually exclusive with [end], [control] aspects.*
`mdc-list-item--with-trailing-radio`	Class *[end] [control]*	Add to allow room for a **trailing** radio button. Can only be used in a **single selection** list. *Mutually exclusive with [end], [control] aspects.*
`mdc-list-item--with-trailing-switch`	Class *[end] [control]*	Add to allow room for a **trailing** switch. **Not yet supported.** Can only be used in a **single selection** list. *Mutually exclusive with [end], [control].*
`role`	Attribute	Requirements: ● Non-interactive: **do not specify** ● Menu item: **menuitem** ● Option item: **option** ● Single selection item: **option** ● Multiple selection item: **option**
`aria-disabled="true"`	Attribute	Must be set for disabled items. Omit if enabled.
`aria-label`	Attribute	Encouraged for a11y.
`tabindex="-1"`	Attribute	Item focus is managed by the component. Items should not be focusable; the controller will alter this programmatically.

Ripple

Path: `ul > li > span[0]`

The ripple should be the item's first child. Non-interactive items should not include a ripple.

Token	Type	Usage
`mdc-list-item__ripple`	Class	Required by the ripple.

Start block

Path: `ul > li > span[1]`

A start block should only be included if the item contains leading content (icon, checkbox, avatar, etc.). It must appear immediately after the ripple within interactive items. Otherwise, it must be the first child.

A selection control or piece of media must appear in the start block. Both are described in their own sections, below.

Token	Type	Usage
`mdc-list-item__start`	Class	Required by the start block.

Content block

Path: `ul > li > span[2]`

The content block is required and must appear between the start and end blocks, if present.

Text should not be added directly to the content block but to the appropriate containers (see subsections).

Token	Type	Usage
`mdc-list-item__content`	Class	Required by the content block.
`id`	Attribute	A leading or trailing control may use `aria-labelledby` or `aria-describedby` to reference the item's content (e.g., as its label). This ID must be applied to the **content** block, **not the item**.

Overline text

Path: `ul > li > span(content) > span[0]`

An optional line of text appearing before all other text. Must be the first child of the content block; must not be the only child. Contains plain text. **Not yet implemented.**

Token	Type	Usage
`mdc-list-item__overline-text`	Class	Required by overline text.

Primary text

Path: `ul > li > span(content) > span[1]`

The list item's primary text; required. Must be the first child or, if overline text is present, the second child. Contains plain text.

Token	Type	Usage
`mdc-list-item__primary-text`	Class	Required by primary text.

Secondary text

Path: `ul > li > span(content) > span[2]`

The list item's secondary text. If specified, must appear directly below primary text as the content block's last child. Contains plain text.

Token	Type	Usage
`mdc-list-item__secondary-text`	Class	Required by secondary text.

End block

Path: `ul > li > span[3]`

An end block should only be included if the item contains trailing content (icon, checkbox, meta text, etc.). It must be the last child.

A single selection control, piece of media, or meta text must appear in the end block. Meta text is included directly (e.g., text is added to the end block). Media and controls are described in their own sections, below.

Token	Type	Usage
`mdc-list-item__end`	Class	Required by the end block.

Media

Media refers to graphical content (including icons, avatars, videos, images, and more) that can be added to an item's start or end block. These blocks (if present) must contain a single media or control child.

Items may contain both leading and trailing media simultaneously.

Icon Media

Path: `ul > li > span(start) | span(end) > i | span | img`

Standard 24 x 24dp Material icons are supported. Fonts, SVG, and images have been tested.

Avatar Media

Path: `ul > li > span(start) | span(end) > span | img`

Images and spans have been tested. Must be 40 x 40dp and suitable for cropping to a circle.

Image Media

Path: `ul > li > span(start) | span(end) > img`

Images have been tested. Must be 56 x 56dp.

Thumbnail Media

Path: `ul > li > span(start) | span(end) > img`

Images have been tested. Must be 40 x 40dp.

Video Media

Path: `ul > li > span(start) | span(end) > img | video | iframe`

Images have been tested; video and iframes work but are not recommended. Must be 100 x 56dp.

Controls

Media refers to selection controls (including radio buttons, checkboxes, and switches) that can be added to an item's start or end block. These blocks (if present) must contain a single media or control child.

Items may only contain a single control, either in the start block or the end block.

Checkbox

Path: `ul > li > span(start) | span(end) > div`

Standard Material checkboxes are supported. Certain attributes are useful when embedded within a list item; these are summarized, below.

Token	Type	Usage
`aria-labelledby="<content block ID>"`	Attribute	Used to associate the control with the item's content block. Set to the content block's ID.
`aria-label`	Attribute	An explicit label; may be used instead of other techniques.
`checked`	Attribute	The item's selection state is established by the control's `checked` attribute, **not** the item's selected class.
`disabled`	Attribute	The item's disabled state is established by the control's `disabled` attribute **and** the item's disabled class.
`name`	Attribute	Useful for distinguishing options.
`value`	Attribute	Useful for distinguishing options.
`tabindex="-1"`	Attribute	The control should not be tabbable; items are selectable, not the controls. Controls should still display a ripple.

Radio Button

Path: `ul > li > span(start) | span(end) > div`

Standard Material radio buttons are supported. Certain attributes are useful when embedded within a list item; these are summarized, below.

Token	Type	Usage
`aria-labelledby="<content block ID>"`	Attribute	Used to associate the control with the item's content block. Set to the content block's ID.
`aria-label`	Attribute	An explicit label; may be used instead of other techniques.
`checked`	Attribute	The item's selection state is determined by the control's `checked` attribute, **not** the item's selected class.
`disabled`	Attribute	The item's disabled state is established by the control's `disabled` attribute **and** the item's disabled class.
`name`	Attribute	Required to establish a radio group (i.e., to link the list's radio buttons together into a single selection group).
`value`	Attribute	Useful for distinguishing options.
`tabindex="-1"`	Attribute	The control should not be tabbable; items are selectable, not the controls. Controls should still display a ripple.

Switch

Path: `ul > li > span(start) | span(end) > div`

Standard Material switches will be supported. **Not yet implemented.**

Auxiliary

Dividers

Path: `ul > li`

A variety of dividers are supported. At most one divider should appear between items. It is not recommended to include a divider at the start or end of a list.

Token	Type	Usage
`mdc-list-divider`	Class	Required by dividers.
`mdc-list-divider--with-leading-padding`	Class *[leading]*	Add to align the divider's leading edge with the item's leading padding. Otherwise, the divider has no leading space. *Mutually exclusive with [leading].*
`mdc-list-divider--with-leading-inset`	Class *[leading]*	Add to align the divider's leading edge with the item's content block. Otherwise, the divider has no leading space. *Mutually exclusive with [leading]; requires one [inset].*
`mdc-list-divider--with-trailing-inset`	Class	Add to align the divider's trailing edge with the item's trailing padding. Otherwise, the divider has no trailing space.
`mdc-list-divider--with-leading-icon`	Class *[Inset]*	Inset the divider's leading edge to account for a leading icon (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-leading-avatar`	Class *[Inset]*	Inset the divider's leading edge to account for a leading avatar (i.e., to visually align with an adjacent item). Must be used in conjunction with

		`mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-leading-thumbnail`	Class *[Inset]*	Inset the divider's leading edge to account for a leading thumbnail (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-leading-image`	Class *[Inset]*	Inset the divider's leading edge to account for a leading image (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-leading-video`	Class *[Inset]*	Inset the divider's leading edge to account for a leading video (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-leading-checkbox`	Class *[Inset]*	Inset the divider's leading edge to account for a leading checkbox (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-leading-radio`	Class *[Inset]*	Inset the divider's leading edge to account for a leading radio button (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`mdc-list-divider--with-lea`	Class	Inset the divider's leading edge to account for

`ding-switch`	*[Inset]*	a leading switch (i.e., to visually align with an adjacent item). Must be used in conjunction with `mdc-list-divider--with-leading-inset`. *Mutually exclusive with [inset].*
`role="separator"`	Attribute	Required by dividers.
`aria-hidden="true"`	Attribute	Add if the divider does not establish logical groups (e.g., because it is purely decorative).