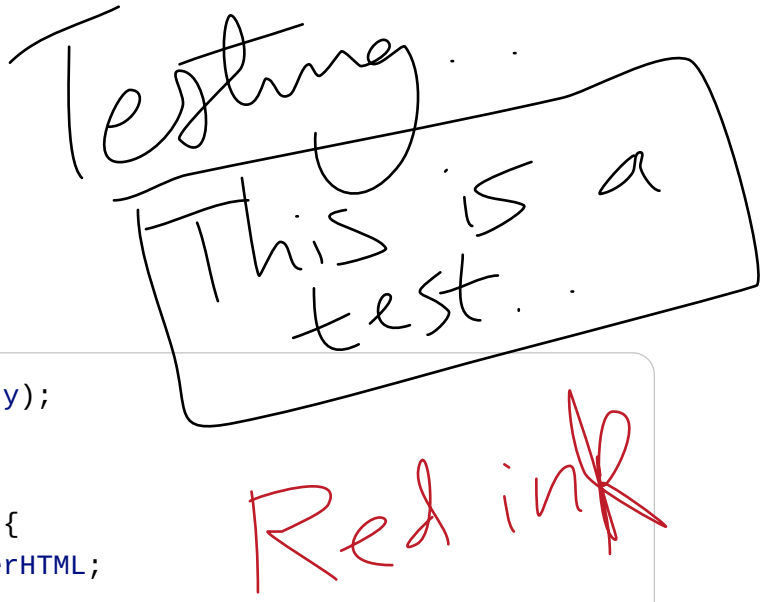
Class Editor



The main entrypoint for the full editor.

Example

To create an editor with a toolbar,

```
const editor = new Editor(document.body);

const toolbar = editor.addToolbar();
toolbar.addActionButton('Save', () => {
  const saveData = editor.toSVG().outerHTML;
  // Do something with saveData...
});
```

See also `docs/example/example.ts`.

Hierarchy

- **Editor**

Defined in Editor.ts:72

INDEX

Constructors

C constructor

Properties

- display
- history
- icons
- image
- notifier
- toolController
- viewport

Methods

- addAndCenterComponents
- addStyleSheet
- addToolbar
- announceForAccessibility
- asyncApplyCommands
- asyncApplyOrUnapplyCommands
- asyncUnapplyCommands
- clearWetInk
- createHTMLOverlay
- dispatch
- dispatchNoAnnounce
- drawWetInk
- estimateBackgroundColor
- focus
- getImportExportRect
- getRootElement
- handleHTMLPointerEvent
- handleKeyEventsFrom
- handlePointerEventsFrom
- hideLoadingWarning
- loadFrom
- loadFromSVG
- queueRerender
- rerender
- sendKeyboardEvent
- sendPenEvent
- setBackgroundColor
- setImportExportRect
- showLoadingWarning
- toDataURL
- toSVG

Text annotation!

More text.

Constructors

¿¿¿¿¿¿???????

constructor

`new Editor(parent, settings?): Editor`

Will this work?

Parameters

- **parent**: *HTMLElement*
- **settings**: *Partial<EditorSettings>* = {}

Returns *Editor*

Example

```
const container = document.body;

// Create an editor
const editor = new Editor(container, {
  // 2e-10 and 1e12 are the default values for minimum/maximum zoom.
  minZoom: 2e-10,
  maxZoom: 1e12,
});

// Add the default toolbar
const toolbar = editor.addToolbar();

// Add a save button
toolbar.addActionButton({
  label: 'Save'
  icon: createSaveIcon(),
}, () => {
  const saveData = editor.toSVG().outerHTML;
  // Do something with saveData
});
```

Defined in Editor.ts:171

Properties

display

```
display: Display
```

Manages drawing surfaces/AbstractRenderers.

Defined in Editor.ts:78

history

```
history: UndoRedoHistory
```

Handles undo/redo.

Example

```
const editor = new Editor(document.body);

// Do something undoable.
// ...

// Undo the last action
editor.history.undo();
```

Defined in Editor.ts:94

Readonly icons

```
icons: IconProvider
```

iconProvider

Defined in Editor.ts:125

Readonly image

```
image: EditorImage
```

Data structure for adding/removing/querying objects in the image.

Example

```
const editor = new Editor(document.body);

// Create a path.
const stroke = new Stroke([
  Path.fromString('M0,0 L30,30 z').toRenderable({ fill: Color4.black }),
]);
const addElementCommand = editor.image.addElement(stroke);

// Add the stroke to the editor
editor.dispatch(addElementCommand);
```

Defined in Editor.ts:113

Readonly notifier

```
notifier: EditorNotifier
```

Global event dispatcher/subscriber.

Defined in Editor.ts:137

Readonly toolController

```
toolController: ToolController
```

Controls the list of tools. See the custom tool example for more.

Defined in Editor.ts:132

Readonly viewport

```
viewport: Viewport
```

Allows transforming the view and querying information about what is currently visible.

Defined in Editor.ts:119

Methods

addAndCenterComponents

```
addAndCenterComponents(components, selectComponents?): Promise<void>
```

Parameters

- **components:** *AbstractComponent*[]
- **selectComponents:** *boolean* = true

Returns *Promise<void>*

Defined in Editor.ts:934

addStyleSheet

```
addStyleSheet(content): HTMLStyleElement
```

Parameters

- **content:** *string*

Returns *HTMLStyleElement*

Defined in Editor.ts:893

addToolbar

```
addToolbar(defaultLayout?): HTMLToolbar
```

Creates a toolbar. If `defaultLayout` is true, default buttons are used.

Parameters

- **defaultLayout:** *boolean* = true

Returns *HTMLToolbar*

a reference to the toolbar.

Defined in Editor.ts:306

announceForAccessibility

```
announceForAccessibility(message): void
```

Announce `message` for screen readers. If `message` is the same as the previous message, it is re-announced.

Parameters

- **message:** *string*

Returns *void*

Defined in Editor.ts:293

asyncApplyCommands

```
asyncApplyCommands(commands, chunkSize): Promise<void>
```

Parameters

- **commands:** *Command*[]
- **chunkSize:** *number*

Returns *Promise<void>*

See

#asyncApplyOrUnapplyCommands

Defined in Editor.ts:760

asyncApplyOrUnapplyCommands

```
asyncApplyOrUnapplyCommands(commands, apply, updateChunkSize): Promise<void>
```

Apply a large transformation in chunks. If `apply` is `false`, the commands are unapplied. Triggers a re-render after each `updateChunkSize`-sized group of commands has been applied.

Parameters

- **commands:** *Command*[]
- **apply:** *boolean*
- **updateChunkSize:** *number*

Returns *Promise<void>*

Defined in Editor.ts:729

asyncUnapplyCommands

```
asyncUnapplyCommands(commands, chunkSize, unapplyInReverseOrder?):
Promise<void>
```

If `unapplyInReverseOrder`, commands are reversed before unapplying.

Parameters

- **commands:** *Command*[]
- **chunkSize:** *number*

- **unapplyInReverseOrder:** *boolean* = false

Returns *Promise<void>*

See

#asyncApplyOrUnapplyCommands

Defined in Editor.ts:766

clearWetInk

```
clearWetInk(): void
```

Clears the wet ink display.

Returns *void*

See

getWetInkRenderer

Defined in Editor.ts:866

createHTMLOverlay

```
createHTMLOverlay(overlay): {
    remove: (() => void);
}
```

Creates an element that will be positioned on top of the dry/wet ink renderers.

This is useful for displaying content on top of the rendered content (e.g. a selection box).

Parameters

- **overlay:** *HTMLElement*

Returns {
  remove: (() => *void*);
}

- **remove**: (() => *void*)

  - ```
    (): void
    ```

    Returns *void*

Defined in Editor.ts:884

dispatch

```
dispatch(command, addToHistory?): void | Promise<void>
```

`apply` a command. `command` will be announced for accessibility.

Parameters

- **command:** *Command*
- **addToHistory:** *boolean* = true

Returns *void* | *Promise<void>*

Defined in Editor.ts:690

dispatchNoAnnounce

```
dispatchNoAnnounce(command, addToHistory?): void | Promise<void>
```

Dispatches a command without announcing it. By default, does not add to history. Use this to show finalized commands that don't need to have `announceForAccessibility` called.

Prefer `command.apply(editor)` for incomplete commands. `dispatchNoAnnounce` may allow clients to listen for the application of commands (e.g. `SerializableCommand`s so they can be sent across the network), while `apply` does not.

Parameters
```

- **command:** *Command*
- **addToHistory:** *boolean* = false

Returns *void* | *Promise<void>*

Example

```
const addToHistory = false;
editor.dispatchNoAnnounce(editor.viewport.zoomTo(someRectangle),
addToHistory);
```

Defined in Editor.ts:712

drawWetInk

```
drawWetInk(...path): void
```

Draws the given path onto the wet ink renderer. The given path will be displayed on top of the main image.

Parameters

- `Rest` ...**path:** *RenderablePathSpec*[]

Returns *void*

See

getWetInkRenderer flatten

Defined in Editor.ts:855

estimateBackgroundColor

```
estimateBackgroundColor(): Color4
```

Returns *Color4*

the average of the colors of all background components. Use this to get the current background color.

Defined in Editor.ts:1113

focus

```
focus(): void
```

Focuses the region used for text input/key commands.

Returns *void*

Defined in Editor.ts:873

getImportExportRect

```
getImportExportRect(): Rect2
```

Returns the size of the visible region of the output SVG

Returns *Rect2*

Defined in Editor.ts:1126

getRootElement

```
getRootElement(): HTMLElement
```

Returns *HTMLElement*

a reference to the editor's container.

Example

```
// Set the editor's height to 500px
editor.getRootElement().style.height = '500px';
```

Defined in Editor.ts:270

handleHTMLPointerEvent

```
handleHTMLPointerEvent(eventType, evt): boolean
```

Dispatches a `PointerEvent` to the editor. The target element for `evt` must have the same top left as the content of the editor.

Parameters

- **eventType:** "*pointercancel*" | "*pointerdown*" | "*pointermove*" | "*pointerup*"
- **evt:** *PointerEvent*

Returns *boolean*

Defined in Editor.ts:430

handleKeyEventsFrom

```
handleKeyEventsFrom(elem): void
```

Adds event listners for keypresses to `elem` and forwards those events to the editor.

Parameters

- **elem:** *HTMLElement*

Returns *void*

Defined in Editor.ts:648

handlePointerEventsFrom

```
handlePointerEventsFrom(elem, filter?): {
    remove: (() => void);
}
```

Forward pointer events from `elem` to this editor. Such that right-click/right-click drag events are also forwarded, `elem`'s contextmenu is disabled.

Parameters

- **elem:** *HTMLElement*
- `Optional` **filter:** *HTMLPointerEventFilter*

Returns {
  remove: (() => *void*);
}

- **remove**: (() => *void*)
  - `(): void`

    Remove all event listeners registered by this function.

    Returns *void*

Example

```
const overlay = document.createElement('div');
editor.createHTMLOverlay(overlay);

// Send all pointer events that don't have the control key pressed
// to the editor.
editor.handlePointerEventsFrom(overlay, (event) => {
  if (event.ctrlKey) {
    return false;
  }
  return true;
});
```

Defined in Editor.ts:604

hideLoadingWarning

`hideLoadingWarning(): void`

Returns *void*

Defined in Editor.ts:281

loadFrom

```
loadFrom(loader): Promise<void>
```

Load editor data from an `ImageLoader` (e.g. an SVGLoader).

Parameters

- **loader**: *ImageLoader*

Returns *Promise<void>*

See

loadFromSVG

Defined in Editor.ts:1044

loadFromSVG

```
loadFromSVG(svgData, sanitize?): Promise<void>
```

Alias for loadFrom(SVGLoader.fromString).

This is particularly useful when accessing a bundled version of the editor, where `SVGLoader.fromString` is unavailable.

Parameters

- **svgData**: *string*
- **sanitize**: *boolean* = false

Returns *Promise<void>*

Defined in Editor.ts:1141

queueRerender

```
queueRerender(): Promise<void>
```

Schedule a re-render for some time in the near future. Does not schedule an additional re-render if a re-render is already queued.

Returns *Promise<void>*

a promise that resolves when re-rendering has completed.

Defined in Editor.ts:793

rerender

```
rerender(showImageBounds?): void
```

Re-renders the entire image.

Parameters

- **showImageBounds:** *boolean* = true

Returns *void*

See

queueRerender

Defined in Editor.ts:821

sendKeyboardEvent

```
sendKeyboardEvent(eventType, key, ctrlKey?, altKey?): void
```

Dispatch a keyboard event to the currently selected tool. Intended for unit testing

Parameters

- **eventType:** *KeyPressEvent* | *KeyUpEvent*
- **key:** *string*
- **ctrlKey:** *boolean* = false

- **altKey:** *boolean* = false

Returns *void*

Defined in Editor.ts:903

sendPenEvent

```
sendPenEvent(eventType, point, allPointers?): void
```

Dispatch a pen event to the currently selected tool. Intended primarially for unit tests.

Parameters

- **eventType:** *PointerDownEvt* | *PointerMoveEvt* | *PointerUpEvt*
- **point:** *Vec3*
- `Optional` **allPointers:** *Pointer*[]

  Deprecated

Returns *void*

Deprecated
See

sendPenEvent sendTouchEvent

Defined in Editor.ts:924

setBackgroundColor

```
setBackgroundColor(color): Command
```

Set the background color of the image.

Parameters

- **color:** *Color4*

Returns *Command*

Defined in Editor.ts:1097

setImportExportRect

```
setImportExportRect(imageRect): Command
```

Resize the output SVG to match `imageRect`.

Parameters

- **imageRect:** *Rect2*

Returns *Command*

Defined in Editor.ts:1131

showLoadingWarning

```
showLoadingWarning(fractionLoaded): void
```

Parameters

- **fractionLoaded:** *number*

  should be a number from 0 to 1, where 1 represents completely loaded.

Returns *void*

Defined in Editor.ts:275

toDataURL

```
toDataURL(format?, outputSize?): string
```

Get a data URL (e.g. as produced by `HTMLCanvasElement::toDataURL`). If `format` is

not `image/png`, a PNG image URL may still be returned (as in the case of `HTMLCanvasElement::toDataURL`).

The export resolution is the same as the size of the drawing canvas.

Parameters

- **format**: "*image/png*" | "*image/jpeg*" | "*image/webp*" = 'image/png'
- `Optional` **outputSize**: *Vec3*

Returns *string*

Defined in Editor.ts:990

toSVG

```
toSVG(): SVGElement
```

Returns *SVGElement*

Defined in Editor.ts:1014