

Design Documentation

BSI CX 22.0

BSI Product Development

Version Date 2024-06-26

Covered Topics

1. Introduction	2
1.1. Getting started	2
1.2. Technical Introduction	2
1.3. Branding & Visual Design	3
1.4. Constraints	4
1.5. Designs since BSI CX 22.0	4
1.6. File format	5
1.7. Content Editor	6
1.8. Structure Reference	7
1.9. Dropzones	8
1.10. Groups	9
2. design.json	11
2.1. The Basics	11
2.2. Content Element Groups	13
2.3. Content Elements	13
2.4. Styles	15
2.5. Configuration capabilities of the Rich Text Editor	16
2.5.1. Feature list	18
2.5.2. Value Lists	20
3. Content Editor	25
3.1. Content editor specific CSS	25
3.2. Predefined Story outlets (Bracket Links)	26
3.2.1. Using anchor links inside a design	28
3.3. Content Elements and Parts	29
3.3.1. Grouping	30
3.3.2. Parts	30
3.4. Data BSI Attributes	48
3.4.1. Control Attributes	48
3.4.2. Form Processor Attributes	49
3.5. Websites	50
3.5.1. Terms	50
3.5.2. Concepts	50
3.5.3. Configuration files	55
3.5.4. Includes	57
3.5.5. Content-Editor	60
3.5.6. Pagination	60
3.5.7. Metadata / Images for the navigation	62
3.5.8. Security - Google reCAPTCHA	63

3.5.9. Creation of websites from existing landingpage templates	65
4. CX Design Build	66
5. Migrate an existing design to the new build	67
5.1. Content Elements	67
5.2. Styles	70
5.3. Design and Preview Templates	71
5.4. Java Script	73
5.5. Stylesheets	73
5.6. Properties	73
5.7. Build Configuration	74
6. Help	76
6.1. Good practices	76
6.1.1. Deleting or deprecating content elements	76
6.1.2. Refactoring content elements	76
6.1.3. Creating accessible designs	76
6.2. Troubleshooting	78
6.2.1. Design Upload	78
6.2.2. Design Update	82
6.2.3. Dropzone Issues	85
6.2.4. Design Creator Issues	86
6.2.5. Content Editor Limitations	86

Wrong CX-version? [Click here](#) to get the documentations of all CX-versions.

1. Introduction

Create experiences that delight customers. For marketing, sales, and service: Find, win, retain, and understand customers with BSI CX. Design, execute and optimize perfect customer journeys. Everything is cross-departmental, individualized for every customer, in real time, across all touchpoints, and fully automated – for extraordinary and emotional experiences.

Learn more about all features of BSI CX by visiting [our product website](#).

TIP

You wan't to build a BSI CX Design? **Learn how to use BSI designs by taking part in the [e-learning for designs](#)!**

NOTE

CX, Customer Experience and Studio. The *BSI Customer Suite* consists of multiple Products, *BSI CX* is one of it. Formerly called *BSI Studio*, *CX* is an abbreviation of *BSI Customer Experience*.

1.1. Getting started

Learn how to use BSI designs by taking part in the [e-learning for designs](#).

The interactive e-learning helps you to reach your goal faster. We have developed this e-learning from years of experience with various customers, with the aim of learning how to work with designs quickly and delivering good results to the customer in a timely manner. In particular, you will learn how to work with our templates so that you don't have to start from scratch.

We recommend to start by adopting an existing design for BSI CX. This will help you to get started and will already come with existing content elements which act as examples for your customizations. BSI provides a set of design templates that are called **Master Templates**. The Master Templates for [email](#) and [web](#) (including landingpage and website) are publicly available on GitHub as an MIT licensed open source project.

Besides, the [design build](#) which is used to build the templates, as well as the library for [email content elements](#) and [web content elements](#) on which the templates are based, are MIT licensed open source projects.

Alternatively, you can start off from scratch by using the [Scaffold Design](#) and following the guidance in this documentation to build a completely new template.

1.2. Technical Introduction

This documentation acts as the primary resource for developers that wish to implement a design for BSI CX. For implementation of designs for BSI CX 22.0 and later, see also [Designs since BSI CX 22.0](#) and [our wiki at GitHub](#).

Designs are a vital part of BSI CX. While focusing on BSI CX Designs, a basic knowledge of the following layers is necessary:



Story

The Story controls the flow of a participant. It offers various steps which can be placed on the storyboard via drag and drop. Your next customer journey delivering a landing page, event invitation, website or newsletter will be born here.

Content

Websites, landing pages and newsletters require rich content. The content editor of BSI CX is here to configure such contents. By user using *Content Elements* – reusable building blocks that can be placed in your content via drag and drop – you can build attractive newsletters, innovative landing pages and feature rich websites.

Design

The foundation of every *Content* is its design. The design defines the available „Content Elements“, the visual representation of the content and the behavior of the content and content elements. The *Design* must be created externally and uploaded into BSI CX. This documentation covers how to build such designs.

1.3. Branding & Visual Design

Each brand has its own and unique visual design language. The branding guidelines of your company or customer define the boundary conditions for building a design. As a design developer for BSI CX, it is your responsibility to get a specification on how the website, landing page or newsletter should look like and what elements it should consist of. Especially when it comes to designing HTML emails, you may have to find a compromise between the developers' ideas of visual design and the actual technical possibilities of HTML emails.

1.4. Constraints

IMPORTANT

Our build and the CX-Editor has to be taken "as-is".

If the Design Validation fails or the Editor doesn't show the Elements properly, it's the responsibility of the developer (probably you) to adjust the template accordingly.

There are some web technologies we do **NOT** support. Two common examples are:

- **Webcomponents** might work in the resulting landingpages, but are not supported within the CX editor.
- **Other Frameworks** like Angular or React are familiar to most web developers, but not supported by CX.

We strongly recommend you to use our npm build or the CX-Design-Creator. Feel free to contact the design team, if you miss a feature or have an improvement idea.

We recommend to start by adopting an existing design for BSI CX. This will help you to get started and will already come with existing content elements which act as examples for your customizations. BSI provides a set of design templates that are called **Master Templates**. The Master Templates for [email](#) and [web](#) (including landingpage and website) are publicly available on GitHub as an MIT licensed open source project (see [Designs since BSI CX 22.0](#)). Alternatively, you can start off from scratch by using the [Scaffold Design](#) and following the guidance in this documentation to build a completely new template.

1.5. Designs since BSI CX 22.0

As of BSI CX 22.0, a new [design build](#) has been introduced to create design ZIP files that are uploaded to BSI CX using a webpack build. The build is NPM based and replaces the previous BSI Design Creator, which used to be available as an executable file. The technical adjustments that come with the new design build are documented in the [design build wiki at GitHub](#). Existing designs can be migrated to the new design build using the migration instructions (see [Migrate an existing design to the new build](#)).

In addition, new Master Templates for email, landingpage and website have been developed. As a new feature, a library is available as a read-only core for these templates. This Standard Library includes all content elements and the Master Template finally builds on this library. Furthermore, new content elements, such as charts, are introduced in the Master Template and more CX features (especially styles) come into play.

Both the Standard Library and the Master Templates are available as MIT licensed open source software on GitHub to enable access to the source code, to enable notifications about new versions and to collaborate on the templates through pull requests and issues:

- Email design: [Master Template email](#) and [Standard Library email](#)
- Landingpage and website design: [Master Template web](#) and [Standard Library web](#)

The current design zip files for uploading to CX are available for download in the GitHub releases

of the respective Master Templates.

1.6. File format

WARNING

Please refer to version 16.3 of the [CX design documentation](#) for file format up to BSI CX version 16.3.

A BSI CX design is stored in a ZIP archive. A valid design for BSI CX since version 22.0 must contain the following mandatory files:

design.html

Defines the design, particularly the dropzones the user can use in the BSI CX content editor to place content elements. The exact design.html syntax is described below. This file must be in the root of the ZIP file.

design.json

Contains metadata on the design, such as names and descriptions of the content elements and groups. The exact design.json syntax is described in chapter [design.json](#). This file must be in the root of the ZIP file.

Besides these two mandatory files your design will need some content elements. Typically, a folder named **content-elements** contains all content elements of the design as **html** files. These elements can later be drag and dropped into the dropzones of the **design.html**. A content element is an HTML snippet that does not require any Doctype, **<html>**, **<head>** or **<body>** tag.

Additional optional files and folders may be supplied in the ZIP file as well. Note that one ZIP file can only contain a single design.

NOTE

The files and folders in the ZIP file must not contain any umlauts.

The following folder structure showcases the content of a simple design:


```
design.zip
|
| design.html ①
| design.json ②
| preview.html ③
|
|-----content-elements ④
|         button.html
|         text.html
|
|-----css
|         styles.min.css
|
|-----img
|         logo.png
|
|-----js
|         3rd-party.min.js
|         scripts.min.js
```

- ① The mandatory file **design.html** defines the base layout of the page/email.
- ② The mandatory file **design.json** contains meta information about the design and its elements.
- ③ The optional file **preview.html** contains a preview of the base layout including some elements.
- ④ The **content-elements** folder holds all content elements **html** files.

1.7. Content Editor

The content editor is where specific websites, newsletters and landing pages are created. The content creator relies on a design of good quality to build contents which on the other hand are referred from a story.

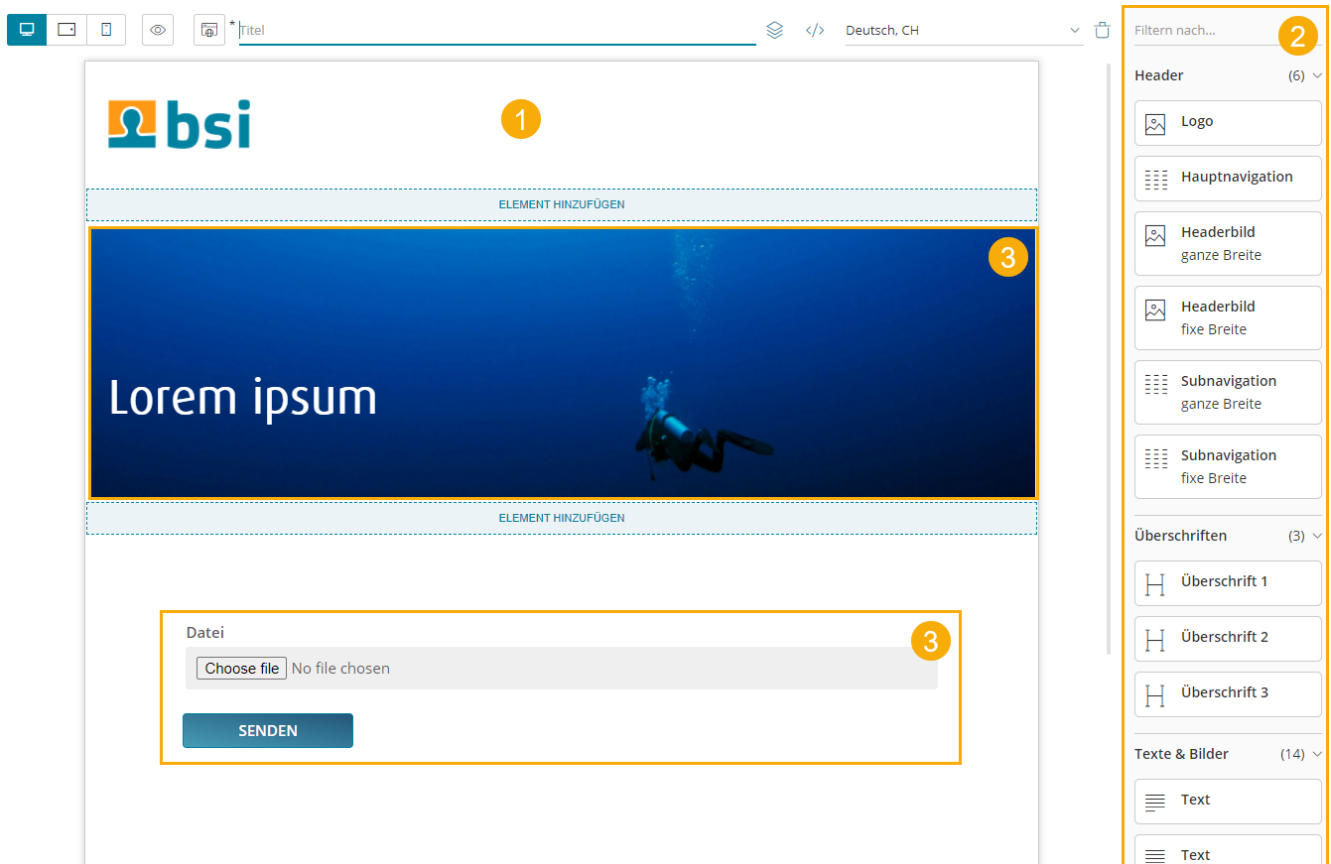


Figure 1. example of the BSI CX content editor

1. Design preview in content editor
2. List of available content elements
3. Use of a customizable content element

1.8. Structure Reference

The file `design.html` contains the `frame` of the design and must include at least one outermost dropzone. It must contain valid, XHTML conform HTML with Doctype, `<html>`, `<head>` and `<body>` tags.

The individual content elements are HTML snippets that are inserted into the content using drag & drop.

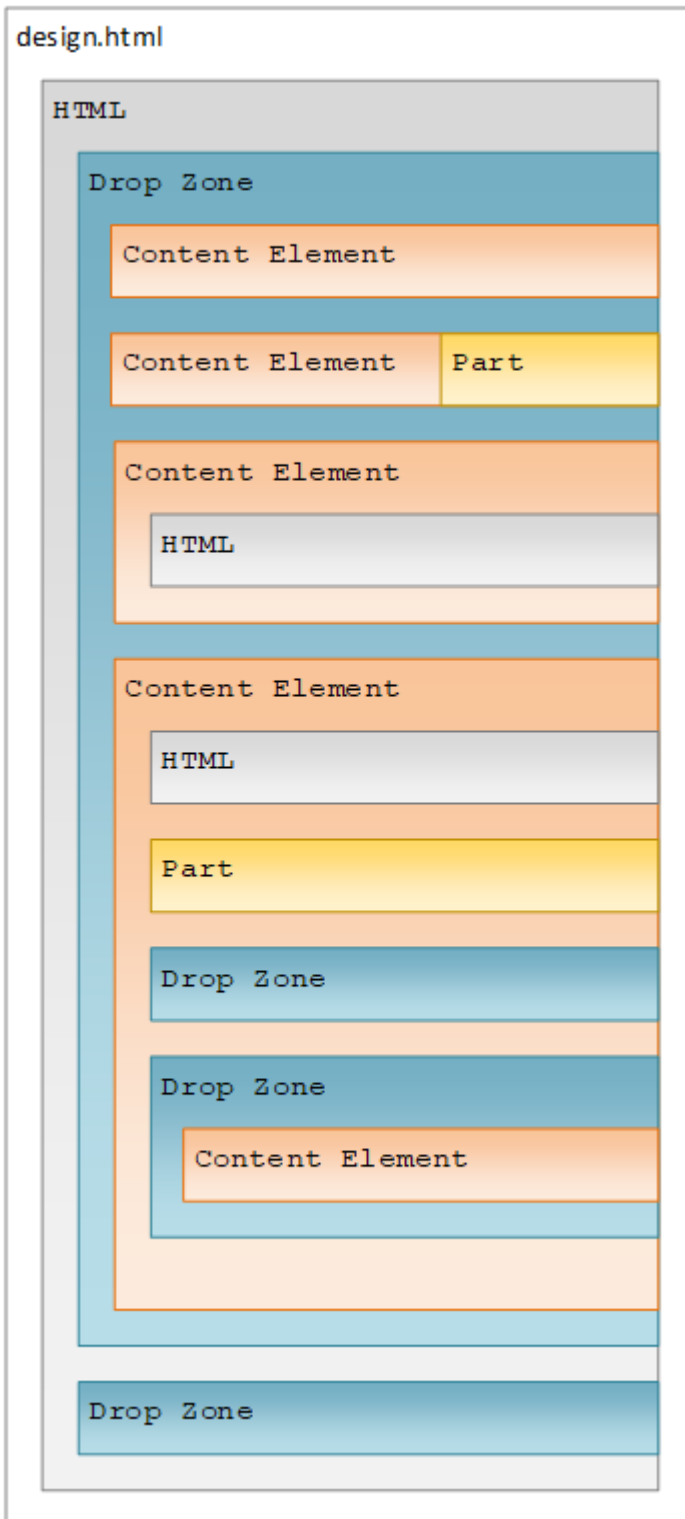


Figure 2. structure reference

1.9. Dropzones

While dragging a content element into the content area, you will notice blue lines and blue areas. These areas are called dropzones and identify areas where certain content elements can be placed. Each dropzone has a list of allowed content elements that are specified in the design. A dropzone is defined by adding the attribute `data-bsi-dropzone` to an HTML container element (e.g. `<div>` or `<table>`). The list of allowed content elements is specified by the `data-bsi-dropzone-allowed-elements` attribute. An optional limit of elements that can be placed inside the dropzone is defined by adding the `data-bsi-dropzone-max-number-of-elements` attribute.

Up to 50% off in January

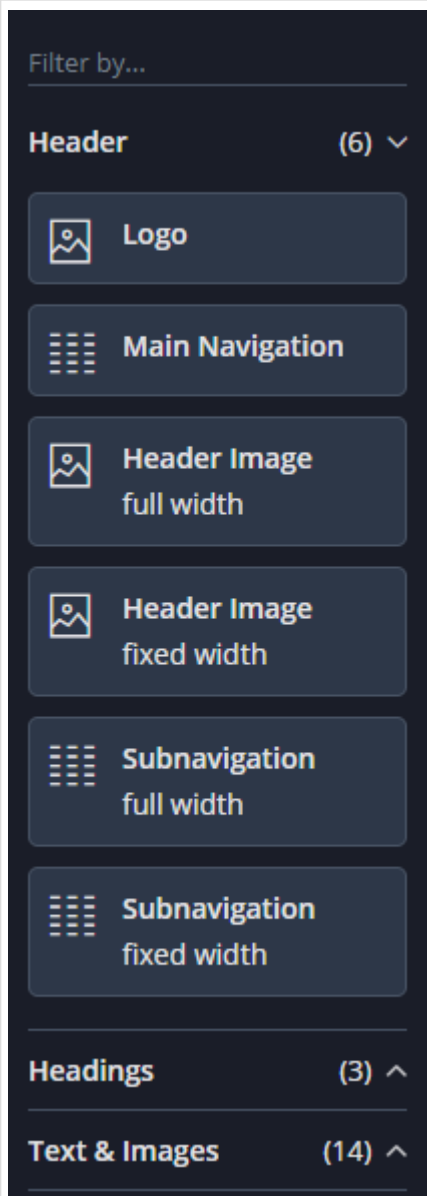
Listing 1. A dropzone that allows to place a maximum of one pre header element inside a newsletter design

```
<div data-bsi-dropzone="preheader-zone" data-bsi-dropzone-allowed-elements="nl-  
preheader" data-bsi-dropzone-max-number-of-elements="1">  
  <!-- Add element -->  
</div>
```

WARNING

Problems can occur when placing a dropzone into a link or button element.
In this case, we recommend using a style.

1.10. Groups



All content elements can be structured in groups. In the example screenshot on the left side, three groups are defined: **Header**, **Headings** and **Text & Images**. The structuring of content elements in such groups is implemented in the **design.json** file. For more information see chapter **Content Element Groups**. A group may be defined only once.

2. design.json

2.1. The Basics

WARNING

BSI CX 22.0 introduced the `design.json` file, which replaces the previously used `design.properties` file. Information on the legacy `design.properties` can be found in the [CX design documentation](#) for CX version 1.3 or earlier.

WARNING

If you are using the [CX Design Build](#), the `design.json` file is generated automatically and no manual adjustments should be made to this file.

Metadata for a design is defined in the `design.json` file. It contains general information on the design, such as the name and author, as well as element specific information, such as with which description and icon the elements are to be displayed to the right of the editor.

Listing 2. Specifying design meta data

```
{
  "title": "My Customer Design",
  "author": "John Doe, Doe Design Agency"
}
```

Group information, content element specific entries, configuration capabilities for styles (allowing you to mixin different CSS styles or adding certain features using JavaScript) as well as instructions on how to customize the rich text editor experience can be found in the subsequent chapters.

Listing 3. Example of a fully featured design.json file

```
{
  "__REM000": "----- Metadata -----",
  "title": "My Customer Design",
  "author": "John Doe, Doe Design Agency",
  "schemaVersion": "22.0",
  "defaultLocale": "en",
  "__REM001": "----- Content-Elements, -Groups and -Parts -----",
  "contentElementGroups": [
    {
      "groupId": "buttons-and-links",
      "label": "Buttons & Links",
      "contentElements": [
        {
          "elementId": "button",
          "label": "Button",
          "icon": "megaphone",
          "file": "content-elements/button.html",
          "parts": [
            {
```

```

        "partId": "link",
        "label": "Button"
    }
],
"styleConfigs": [
    "background-color"
]
}
]
},
{
    "groupId": "text-and-images",
    "label": "Text & Images",
    "contentElements": [
        {
            "elementId": "text",
            "label": "Text",
            "icon": "text",
            "file": "content-elements/text.html",
            "parts": [
                {
                    "partId": "formatted-text",
                    "label": "Text",
                    "htmlEditorConfig": "custom"
                }
            ]
        }
    ]
}
],
"__REM002": "----- Style Configs -----",
"styleConfigs": {
    "background-color": {
        "label": "Background color",
        "cssClasses": [
            {
                "cssClass": "green-background",
                "label": "Green"
            },
            {
                "cssClass": "red-background",
                "label": "Red"
            }
        ]
    }
},
"__REM003": "----- HTML Editor Configs -----",
"htmlEditorConfigs": {
    "custom": {
        "features": [

```

```

        "bold",
        "italic",
        "underline",
        "textColor"
    ],
    "textColors": [
        "#16616d",
        "#383e42"
    ]
  }
}
}

```

2.2. Content Element Groups

Each content element is assigned to a group. Add an entry to the `design.json` file to specify an identifier and a display name for each *group* element.

Listing 4. Giving groups clearly identifiable descriptions

```

"contentElementGroups": [
  {
    "groupId": "buttons-and-links",
    "label": "Buttons & Links"
  },
  {
    "groupId": "text-and-images",
    "label": "Text & Images"
  }
]

```

2.3. Content Elements

For each content element the following entries should be defined in order to get a clean user experience. The *element identifier* is the value of the `data-bsi-element` attribute, that is defined in the corresponding `html` file.

Listing 5. Important metadata for content elements

```
"contentElements": [
  {
    "elementId": "logo",
    "label": "Cooperate Logo",
    "description": "Visualize your corporate logo",
    "icon": "image",
    "file": "content-elements/logo.html",
    "parts": [
      {
        "partId": "image",
        "label": "Mobile image for Logo"
      }
    ]
  }
]
```

Refer to the following illustration to pick a value for *icon*.

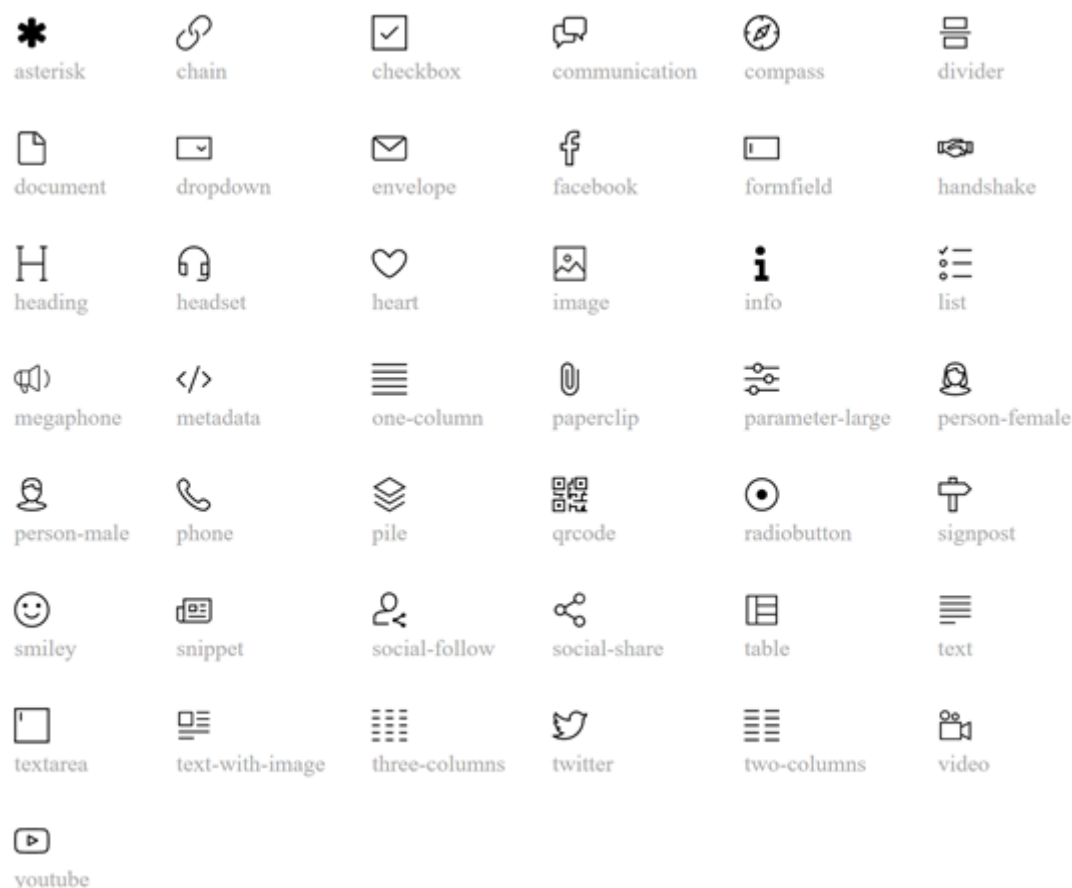


Figure 3. icon identifier reference

For element parts, labels can be defined. Those are then used as section headings while editing a content element in the content editor. *partId* is the *data-bsi-element-part* value of the element part.

2.4. Styles

WARNING

We strongly discourage from using styles in email templates. Many email clients still require certain styles to be set through inline styles which does not work well with styles on element level as it is done by the *styles feature* in this release. With CX 23.2 an improved *styles feature* supporting inline styles was introduced to be used in email templates. For more information visit the [CX design documentation](#) for CX version 23.2 or later.

By defining styles for a certain content element, its characteristics can be controlled. By using styles to make a content element configurable, it is not necessary to implement an additional content element for an almost identical content element. A requirement to implement a button in two different colors (red and green) would therefore be implemented by creating a *button* content element with two styles: red and green. Styles are only defined once, and each element can have 0 to n style capabilities. Each style capability will then be transformed to a dropdown in the editing dialog of a content element, where the specific style can be applied to a content element.



Figure 4. example of a style dropdown

Listing 6. Defining style capabilities

```
"styleConfigs": {
  "background-color": {
    "label": "Background color",
    "cssClasses": [
      {
        "cssClass": "green-background",
        "label": "Green"
      },
      {
        "cssClass": "red-background",
        "label": "Red"
      }
    ]
  }
}
```

1 to n styles can be defined to appear as individual dropdowns in the editor, where one style can be selected from the list of styles.

Listing 7. Assigning style capabilities to content elements

```
"contentElements": [
  {
    "elementId": "button",
    "label": "Button",
    "styleConfigs": [
      "background-color",
      "border-color",
      "text-color"
    ]
  }
]
```

To ensure that a style is effective, a CSS class must be defined for it. If a style is preselected on an element, it is sufficient to add the desired CSS class to the respective element in the HTML.

Listing 8. CSS class for a style

```
.colored-button.green-background { background-color: green; }
.colored-button.red-background { background-color: red; }
```

Listing 9. how styles are applied on content elements

```
<button data-bsi-element="button" class="colored-button green-background"></button>
```

2.5. Configuration capabilities of the Rich Text Editor

By defining a content element part of type `formatted-text`, a fully featured rich text editor will be available to the users of the content editor. The built-in WYSIWYG rich text editor is powered by [Froala](#). BSI allows to customize a subset of Froala features directly out of a design. In order to do so, a configuration section in the `design.json` can be used to customize the features of the rich text editor. Each element part of type `formatted-text` can have its own custom list of features, although in most cases one feature definition is shared among different content elements.

A typical, simple rich text editor configuration will look like this:

Listing 10. Rich text editor configuration example

```
"htmlEditorConfigs": {
  "my-config": {
    "features": [
      "bold",
      "italic",
      "underline",
      "textColor"
    ],
    "textColors": [
      "#ff00cc",
      "#aabbcc"
    ]
  }
}
```

The above configuration (named `my-config`) needs to be assigned to a concrete element part in order to apply the customizations:

Listing 11. Applying an HTML editor configuration on a content element 'text'

```
"contentElements": [
  {
    "elementId": "text",
    "label": "Text",
    "parts": [
      {
        "partId": "formatted-text",
        "label": "Text",
        "htmlEditorConfig": "my-config"
      }
    ]
  }
]
```



The entire list of supported configuration options is documented in the subsequent chapters.

2.5.1. Feature list

The most important configuration option is the list of features that will be shown in the WYSIWYG editor.

If one wants to support **bold**, *italic*, and underlined text, the definition for doing so would be like this:

Listing 12. Feature list example

```
"htmlEditorConfigs": {
  "my-config": {
    "features": [
      "bold",
      "italic",
      "underline"
    ]
  }
}
```

Each key in the list below (indicated in **bold**) identifies a certain feature, which is then displayed as a separate button in the editor. The sequence is irrelevant and does not affect the display order, as the display order is given by the server-side implementation.

The following features are available:

bold	the text can be formatted in bold .
italic	the text can be formatted in <i>italics</i> .
underline	the text can be <u>underlined</u> .
strikeThrough	the text can be displayed with a line through the center.
subscript	the text can be displayed as a _{subscript} .
superscript	the text can be displayed as a ^{superscript} .
fontSize	the font size can be changed based on the <i>font-sizes</i> value list.
lineHeight	the line height can be changed based on the <i>line-heights</i> value list.
textColor	the text color can be changed based on the <i>text-colors</i> value list.

backgroundColor	the background color can be changed based on the <i>background-colors</i> value list.
alignLeft	the text can be left aligned.
alignCenter	the text can be centered.
alignRight	the text can be right aligned.
alignJustify	the text can be justified (As most browsers are terrible in handling justified text properly, we recommend to not enable this feature).
formatOL	the text can be displayed as an organized list.
formatUL	the text can be displayed as an unorganized list.
outdent	the text can be outdented (useful for sub-lists).
indent	the text can be indented (useful for sub-lists).
paragraphFormat	changes the format of the paragraph based on the <i>formats</i> value list.
quote	display the text as a quote.
specialCharacters	inserts special characters.
emoticons	inserts emoji.
insertLink	permits the immediate insertion of links without having to use a wildcard.
html	displays and edit the HTML of the text.
help	displays Help.

In addition, the following features are **always displayed in the editor** and must therefore not be listed separately:

clearFormatting	clears all formatting from the selected text.
undo	undoes the change.
redo	redoes the change.

fullscreen displays the editor in full screen mode.

selectAll selects the entire text in the editor.

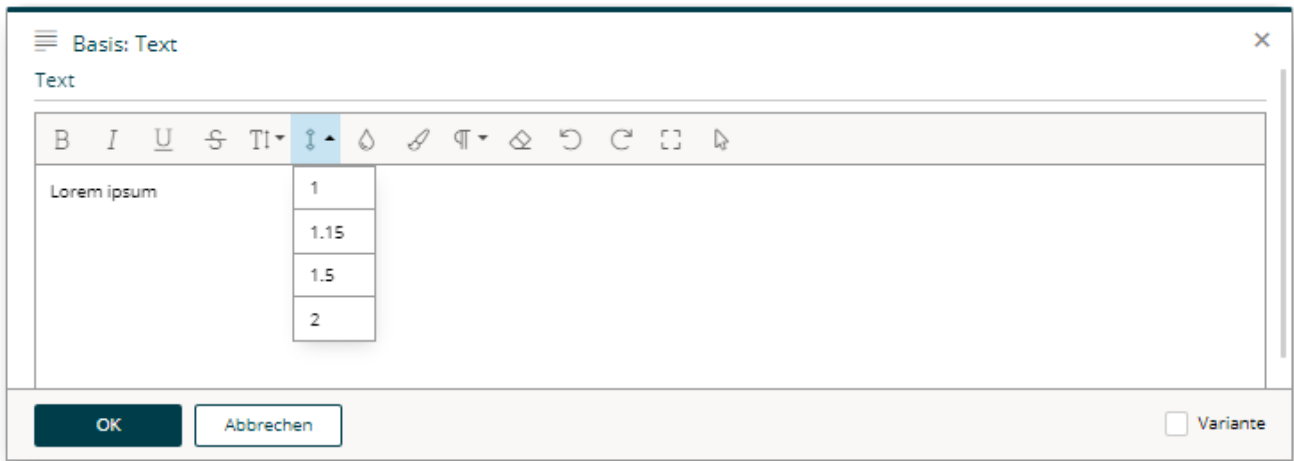
Our experience has shown that these features are useful in the vast majority of cases and that the editing experience would be significantly affected without them.

2.5.2. Value Lists

A list of values that will be shown in the editor can be defined for certain features (e.g. the list of colors when the font color feature is active).

Listing 13. An example using configuration with different value lists set

```
"htmlEditorConfigs": {  
  "my-extended-config": {  
    "features": [  
      "bold",  
      "italic",  
      "underline"  
    ],  
    "textColors": [  
      "#16616d",  
      "#ff7d00",  
      "#383e42"  
    ],  
    "backgroundColors": [  
      "#ffffff",  
      "#383e42"  
    ],  
    "formats": [  
      "p",  
      "h1",  
      "h2",  
      "pre"  
    ],  
    "fontSizes": [  
      12,  
      16,  
      24  
    ],  
    "fontSizeUnit": "px",  
    "fontSizeDefault": 16,  
    "lineHeights": [  
      1,  
      1.15,  
      1.5,  
      2  
    ],  
    "enter": "p"  
  }  
}
```

2.5.2.1. Colors

To customize the color picker for `textColor` and `backgroundColor`, provide a list of colors as follows:

Listing 14. textColor example

```
"textColors": [
  "#16616d",
  "#ff7d00",
  "#383e42"
]
```

Listing 15. backgroundColor example

```
"backgroundColors": [
  "#ffffff",
  "#383e42"
]
```

2.5.2.2. Paragraph formats

Paragraph formatting and headings can be configured as well. This may be customized through the `formats` value list, which will only have an effect if the feature `paragraphFormat` is active.

Listing 16. formats example

```
"formats": [
  "p",
  "h1",
  "h2",
  "pre"
]
```

The following formats can be provided:

p for an html paragraph (<p>)

h1 for an html H1 title (<h1>)

h2 <h2>

h3 <h3>

h4 <h4>

h5 <h5>

h6 <h6>

pre for preformatted text (<pre>)

2.5.2.3. Line heights

To customize the **lineHeight** value list, provide a list of heights in percentages, comma separated. The line height values define the factor by which factor the current line height is multiplied based on the font size used.

Listing 17. lineHeight example

```
"lineHeights": [  
  1,  
  1.15,  
  1.5,  
  2  
]
```

2.5.2.4. Font size

To customize the list of font sizes, provide a list of sizes for **fontSizes**. A custom default value can be set by using **fontSizeDefault**.

Size information is provided numerically. If several sizes are permitted, then each individual entry is listed in a comma separated format. The data on font sizes (font size and font size default) are absolute values.

The unit of measurement (**fontSizeUnit**) defines which unit to be used for information regarding the font sizes. The following are permitted:

px Pixels

em Relative to font size of the parent

rem	Relative to font size of the root element
pt	Points
cm	Centimeters
mm	Millimeters

Listing 18. fontSizes, fontSizeUnit and fontSizeDefault example

```
"fontSizes": [
  8,
  12,
  16,
  24,
  32
],
"fontSizeUnit": "px",
"fontSizeDefault": 16
```

2.5.2.5. Entry mode

By defining the entry mode (enter), you define what happens in the editor when the `Enter` key on your keyboard is pressed. The following options are available:

- p** to enclose the text with a paragraph (`<p>`)
- div** to enclose the text with a `<div>` tag
- br** to simply insert a `
` tag for a line break

In **p** and **div** modes, there is the option to force a simple line break with `Shift + Enter`. This will add `
` without immediately ending the block and starting a new one.

Listing 19. enter mode example

```
"enter": "div"
```

3. Content Editor

3.1. Content editor specific CSS

IMPORTANT

CX 1.2.46

Sometimes it is useful to apply special CSS rules to the content when the page is in edit or preview mode. There are two special CSS classes set on the document's `<html>` tag to address such use cases:

- **bsi-ce-edit-mode** is set when the document is shown in the content editor.
- **bsi-ce-preview-mode** is set when the document is shown in preview mode inside the content editor.

The preview mode outside the content editor or any content that is delivered by a story will not have any of the above listed classes.

Listing 20. editor specific CSS example: HTML

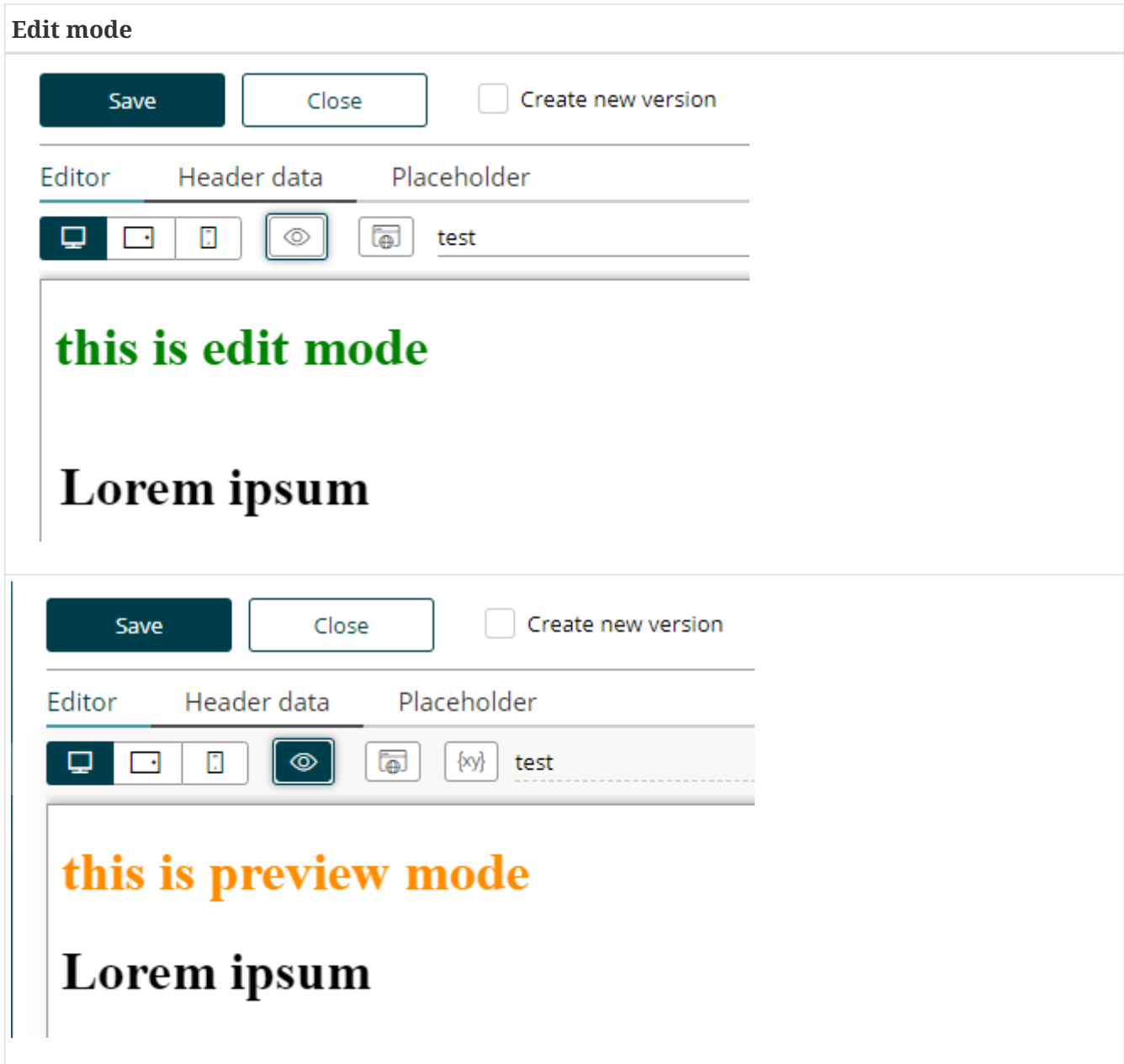
```
<html><head><!-- head omitted --></head>
<body>
  <h1 class="hide show-if-edit">this is edit mode</h1>
  <h1 class="hide show-if-preview">this is preview mode</h1>
  <!-- rest of body omitted -->
</body>
```

Listing 21. editor specific CSS example: CSS

```
.hide { display: none; }
.bsi-ce-edit-mode .show-if-edit { display: block; color: green; }
.bsi-ce-preview-mode .show-if-preview { display: block; color: darkorange; }
```

Table 1. editor specific CSS example: resulting representation in the content editor

Edit mode
Preview Mode



3.2. Predefined Story outlets (Bracket Links)

One way to define a link out of a design is to use a `data-bsi-element-part="link"` annotation on a link (`<a>`) element. Doing so will add a link editor to enter a URL and a link text to the content editor.

Listing 22. Adding an editable link to a content element.

```
<a data-bsi-element-part="link" href="https://www.bsi-software.com/">BSI Website</a>
```

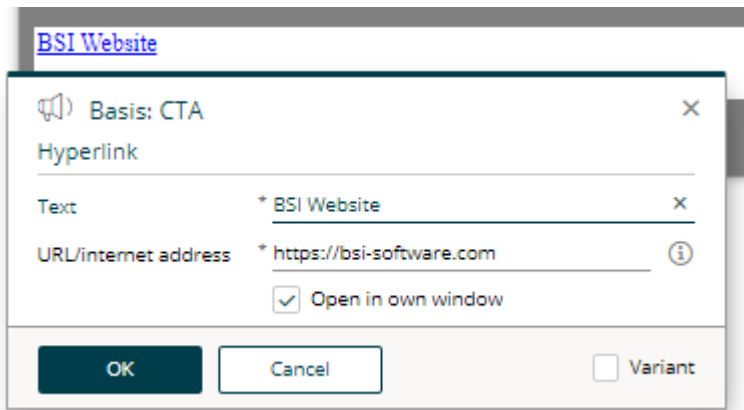


Figure 5. visualization of an editable link

Another way to define a link directly out of a design is by using the CX *bracket link syntax* for links. This mechanism is built into the content editor itself to create a link anywhere in a text.

Listing 23. Example on adding a link anywhere in the text.

```
<span data-bsi-element-part="plain-text" >If you do not wish to receive further
promotions, you can [unsubscribe] now.</span>
```



1. Definition of links using the CX bracket syntax.
2. Each link is converted into a story outlet. Each outlet can then be linked to an individual action in the story.
3. The content that is delivered to the user.

While rendering the above content, CX will convert the `[unsubscribe]` term to a link tag internally, create a unique hyperlink and will add a new outlet to the step to which the current content belongs. When using bracket links, the story designer will see the provided link as a story outlet by default. This allows the Story Designer to link the click action on the link with other steps (e.g. a subsequent landing page step). In the example above, the *Email Step* will already have an outlet named *unsubscribe*. The bracket technique itself is usable everywhere in the content editor and is not limited to the usage in a design.

The convenience format of a bracket link for CX is `[link text|ID]`. The full syntax is `[ID|text=link text|url=https://example.com|newWindow]`.

You may use the following optional parameters:

- `url` can be used to explicitly specify an absolute URL.
- `newWindow` will force to open the target in a new tab or window.
- `follow` removes the `nofollow` value from the html `rel` attribute. Without `nofollow`, bots are allowed to crawl your page (not recommended for personalised links).
- `login` creates a link having the query parameter `?login`, which redirects to the login page in a website.
- `logout` creates a link having the query parameter `?logout`, which removes the authenticated user from a CX HTTP session in a website.

If a bracket itself should be part of a link, it can be escaped with a backslash (e.g. `[\[click me\]|id]` will then be visualized as `[click me]`).

You can also place a bracket link anywhere in the design. An example of doing so is to add it to an `href` of a link.

Listing 24. Adding a bracket link to a link will allow us to track the "Facebook" link in CX.

```
<a href="[Facebook]"></a>
```

WARNING

Be reminded that all CX will replace everything that looks like a bracket link. It is your responsibility as a developer to use escaping where brackets have a different meaning.

Listing 25. Escaping is important in cases like this one.

```
<!-- CX will convert this to a bracket link -->
<input pattern="[0-9]">

<!-- escaping is mandatory for the pattern mechanism to work as desired -->
<input pattern="\[0-9]">
```

3.2.1. Using anchor links inside a design

First of all, it is possible to use anchor links in the form of `#top` inside a CX design. Anchor links are allowed as targets of `link parts`. However, it is not possible to use anchor links as `url` of a bracket link. This is due to the fact, that bracket links meant to be trackable by CX and this requires a complete roundtrip to the server.

```
<section id="news">
  <!-- some news -->
</section>

<!-- some more html code -->

<div data-bsi-element="cta">
  <a href="#news" data-bsi-element-part="link">Jump to news</a>
</div>
```

3.3. Content Elements and Parts

This chapter describes the structure of the `content-elements.html` file and the available content elements. The `content-elements.html` file contains HTML snippets for the content elements that can be selected in BSI CX. To do so, BSI CX searches for elements with the `data-bsi-element` attribute in the HTML code. The recommendation is to define all content elements at the same level as the DOM, for example, as `<div>` directly below the `<body>` element. However, for better visualization during development, the developer can also create additional elements (e.g., with CSS borders or margins) around the elements marked with `data-bsi-element`. BSI CX will ignore those, though.

TIP

It is not necessary for the `content-elements.html` file to have a `<html>`, `<head>` or `<body>` tag. But the containing HTML must be valid.

The following listing illustrates a very simple content element:

```
<div data-bsi-element="logo">
  
</div>
```

As we can see in the listing, `data-bsi-element` defines the identifier of the content element. It must not contain any spaces. The identifier corresponds to the value that can be used in the attribute called `data-bsi-dropzone-allowed-elements` in the dropzone. The element marked with the identifier, when used, is inserted into the relevant dropzone (not only the child elements).

WARNING

The content editor specifically highlights the content elements. To make sure this works properly, all content elements must be block elements rather than inline elements.

A content element can define one or more internal dropzones. The definition is the same as for `design.html`. For example, use it with a content element for a 3-column layout with three internal dropzones. While the dropzones may be nested arbitrarily, please pay attention to usability.

3.3.1. Grouping

Complex designs can consist of many elements. For better organization, it is mandatory to group the elements. To do so, the `data-bsi-group` attribute can be used. The following listing illustrates such a grouping:

```
<div data-bsi-group="texts">
  <div data-bsi-element="default-text" data-bsi-element-part="formatted-text">
    <p>Lorem ipsum</p>
    <ul>
      <li>lorem</li>
      <li>ipsum</li>
    </ul>
  </div>
  <div data-bsi-element="raw-text" data-bsi-element-part="plain-text">Lorem
ipsum</div>
</div>
<div data-bsi-group="images">
  <div data-bsi-element="full-size-image">
    
  </div>
</div>
```

3.3.2. Parts

A content element part references, inside a content element, an area (such as text or image) that can be edited directly with the BSI CX content editor. The content element "text with image" knows, for example, the two areas `image` and `plain-text`. The identifiers must be adopted exactly as they are. The following listing illustrates the example:

```
<figure class="element text-with-image" data-bsi-element="text-with-image"> ①
   ②
  <figcaption class="text" data-bsi-element-part="plain-text">Lorem
ipsum</figcaption> ③
</figure>
```

- ① This is the content element definition.
- ② The `image` part defines an editable image.
- ③ The `plain-text` part defines an editable plain text.

TIP

The individual parts are likely to contain sample content already. This content must fit in terms of the structure (e.g., only plain text with `plain-text`, and a correct table definition with `table`, etc.)

3.3.2.1. Plain Text

The part `plain-text` can be applied to any HTML element that can contain text. The text is inserted

within the element. The element must not have children with other content element parts (content is removed). A single-line text field without formatting is available as the editor. When using the optional attribute `data-bsi-multiline`, the text field becomes multiline.

IMPORTANT

Boolean attributes like `data-bsi-multiline` require an empty value `=""` or sample value like `"true"`.

Listing 27. Sample usage of the plain text part.

```
<h1 data-bsi-element="title-h1" data-bsi-element-part="plain-text" class="element title-h1">Lorem ipsum</h1>
```

Listing 28. Sample usage of the plain text part with the multiline feature.

```
<p data-bsi-element="paragraph" data-bsi-element-part="plain-text" data-bsi-multiline="">Lorem ipsum<br>dolor sit amet</p>
```

3.3.2.2. Formatted Text

The part `formatted-text` can be applied to any HTML element that can contain text. The element must not have children with other content element parts (content is removed). The rich text field with a limited number of formatting options is available as an editor. For more information about the available configuration options see chapter [Configuration capabilities of the Rich Text Editor](#).

Listing 29. Sample usage of the formatted text part.

```
<section data-bsi-element="section" data-bsi-element-part="formatted-text">
  <h1>Lorem ipsum dolor sit amet</h1>
  <p>Proin porta <strong>pharetra est non</strong> molestie.</p>
  <p><a href="https://www.example.org/">More</a></p>
</section>
```

3.3.2.3. HTML

The `html` part can be applied to any HTML element. The code is inserted within the element. The element must not have children with other content element parts (content is removed). A text field is used as the editor. HTML can be written directly into this field.

Listing 30. Sample usage of the HTML part.

```
<div data-bsi-element="html" data-bsi-element-part="html">
  <iframe src="https://www.openstreetmap.org/export/embed.html" style="height: 500px;"></iframe>
</div>
```

This part can be useful for:

- Prototyping

- Quick-and-Dirty Hacks
- Embedding any HTML code

WARNING

It is not allowed to place any `data-bsi-element` or `data-bsi-element-part` attributes inside the HTML code. This won't work as expected and leads to invalid content.

3.3.2.4. Video

The `video` part can be applied to any block level HTML element. This part can be used to display information and thumbnails of external videos. It can also be used to embed external videos. Within the element, various elements can be used as placeholders:

- Each `<iframe>` element is used to embed an external video player.
- Each `` element with a `data-bsi-video-thumbnail` attribute is used to display the video thumbnail.
- Each HTML element with the `data-bsi-video-title` attribute is used to display the video title.
- Each HTML element with the `data-bsi-video-description` is used to display the video description.
- Each `<a>` element with the `data-bsi-video-link` is used to place a link to the external video. The text of the hyperlink will not be touched, only the `href` attribute will be filled.

IMPORTANT

Fetching the video's title and description may requires an active API key for the appropriate provider.

IMPORTANT

Boolean attributes like `data-bsi-video-title` require an empty value `=""` or sample value like `="true"`.

The following external video providers are supported:

YouTube

Without a configured API key, only the thumbnails are automatically picked. More information on obtaining a YouTube API key: <https://developers.google.com/youtube/v3/getting-started>

Be aware that there are two kinds of URLs involved: One is the link to the video platform itself for the `href` attribute on the `<a>` tag and there is the embedded URL for the `<iframe>` player. In the case of YouTube <https://www.youtube.com/watch?v=YpnFs4aBLQA> is the link to the video platform. The corresponding embedded URL would be <https://www.youtube.com/embed/YpnFs4aBLQA>. If you want to provide sample content for your element you may have to fill both URLs in the right tag. The following listing illustrates the use of sample content for a simple video element:

Listing 31. Example of a simple video element.

```
<div class="element video" data-bsi-element="video-simple">
  <div class="responsive-video" data-bsi-element-part="video" data-bsi-video-data-
link="https://www.youtube.com/watch?v=YpnFs4aBLQA"> ①
    <iframe width="1120" height="630" frameborder="0"
src="https://www.youtube.com/embed/YpnFs4aBLQA" allow="autoplay; encrypted-media"
webkitallowfullscreen mozallowfullscreen allowfullscreen></iframe> ②
  </div>
</div>
```

- ① The link to the video platform is placed inside the `data-bsi-video-data-link` attribute.
- ② The embed video URL is placed inside the `src` attribute of the `<iframe>` tag.

This results in the following content element:

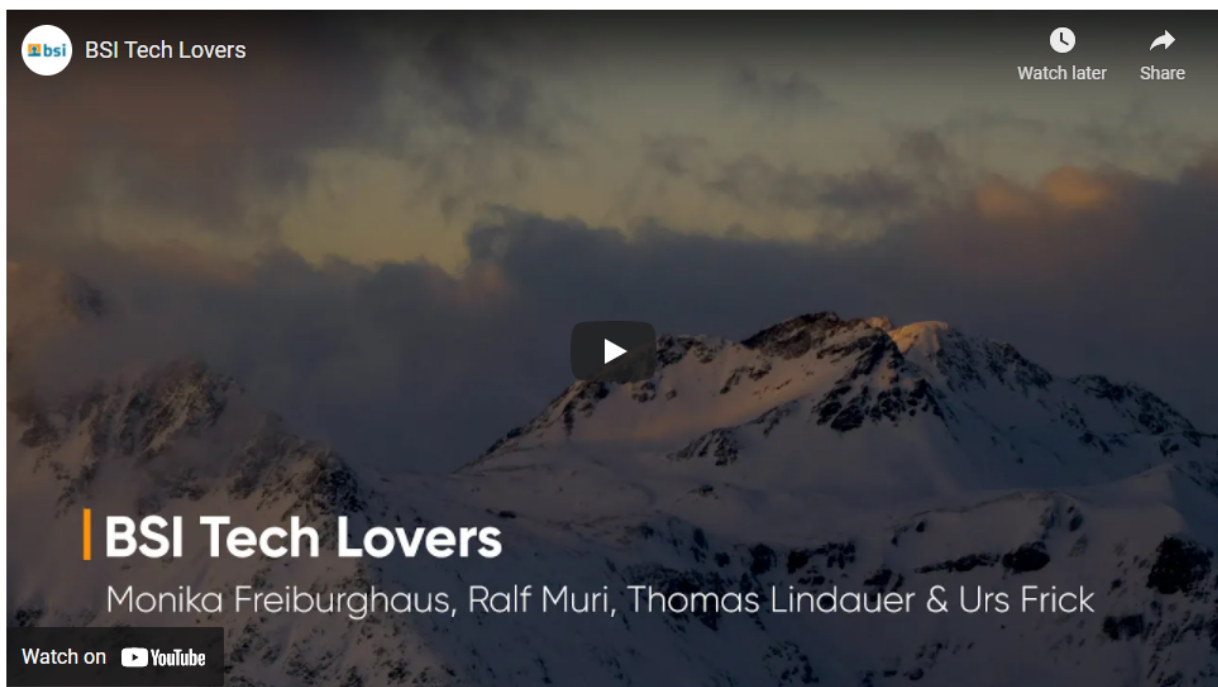


Figure 6. Simple video element in action.

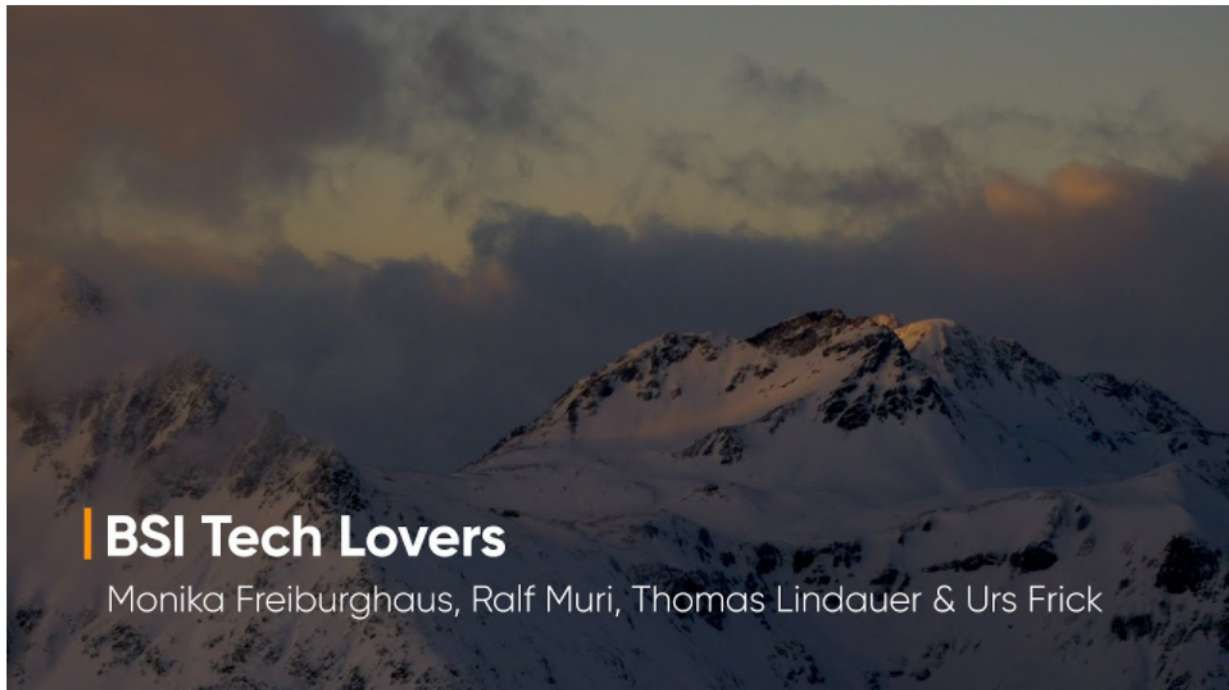
A more complex video element would be the following:

Listing 32. Example of a complex video element.

```
<div class="element video" data-bsi-element="video-thumbnail">
  <div class="video-thumbnail" data-bsi-element-part="video" data-bsi-video-data-
link="https://www.youtube.com/watch?v=YpnFs4aBLQA"> ①
    <p>
      <a href="https://www.youtube.com/watch?v=YpnFs4aBLQA" data-bsi-video-
link="" target="_blank"> ②
         ③
      </a>
    </p>
    <h3 data-bsi-video-title="">BSI Tech Lovers</h3> ④
    <p data-bsi-video-description="">Für mehr WOW-Momente mit BSI. Monika
Freiburghaus, Ralf Muri, Thomas Lindauer und Urs Frick zeigen Einblicke in ihre
persönlichen WOW-Momente.</p> ⑤
  </div>
</div>
```

- ① The link to the video platform is placed inside the `data-bsi-video-data-link` attribute.
- ② The link to the video platform is also placed inside the `href` attribute.
- ③ The placeholder image can also be part of the design, it is not mandatory to provide the URL to the real image here.
- ④ The content of the first tag marked with `data-bsi-video-title` will be used as placeholder for the title.
- ⑤ The content of the first tag marked with `data-bsi-video-description` will be used as placeholder for the video description.

This results in the following content element:



BSI Tech Lovers

Für mehr WOW-Momente mit BSI. Monika Freiburghaus, Ralf Muri, Thomas Lindauer und Urs Frick zeigen Einblicke in ihre persönlichen WOW-Momente.

Figure 7. Complex video element in action.

The video part is quite powerful and can be used in various ways:

- Enrich your landing page or website with embedded videos.
- Use video thumbnails in your content.
- Send emails with link, thumbnail, title and description of an external video.

3.3.2.5. Image

The **image** element part can only be placed inside of a `` or `<a>` tag. The source is inserted in the **src** attribute.

Listing 33. Example of an image element

```
<div data-bsi-element="image-simple">
  
</div>
```

It is not possible to omit the image of an **image** part. Therefore it is mandatory to specify an image in the content editor. Otherwise a validation error will occur.

srcset

By specifying the **srcset** attribute, the images are also scaled accordingly. If a **srcset** attribute is present, the **src** attribute is set to the image at its highest resolution (according to the defined **srcset**). If there is no **srcset** attribute, the selected image is not scaled and is placed directly as **src**. For images with dynamic sources (the image is set through the story), the definition of the **srcset** is

ignored. If you set an image in the content editor, make sure it is of decent size. After saving the content, the image is scaled to the sizes specified in the `srcset` attribute and stored on the CX resource server. Depending on the resolution of device that is requesting the landing page/website, a different resolution of the image is loaded.

TIP We highly recommend to set the `srcset` attribute.

Listing 34. Example of a image element that scales to the sizes specified in `srcset`

```
<div data-bsi-element="image-simple">
  
</div>
```

The attribute `data-bsi-hide-link` controls whether the field *URL/internet address* and the checkbox *Open in new window* is visible and editable in the content editor. A design developer may still wrap the image inside an `<a>` tag in the content element, but the user cannot change the link.

IMPORTANT The attribute `data-bsi-hide-link` is available from CX 1.3.40 onward

Providing an image through the story

Dynamic image sources can be specified with angle brackets, e.g. `<my-dynamic-image>`. The image URL will then be provided by the story. Please note that in this case the `srcset` attribute will be ignored. Thus, dynamic image sources are not scaled.

3.3.2.6. background-image

An arbitrary element which permits the use of a CSS `background-image`. It places the CSS attribute `background-image`. The image selection is available as the editor just like with the image part.

Listing 35. Example usage of the background-image part.

```
<div class="element scene" data-bsi-element="scene">
  <div data-bsi-element-part="background-image" style="background-image:
  url('example.jpg')"></div>
</div>
```

3.3.2.7. table

A `<div>` oder `<td>` element with a table element as a direct child element. The table content is filled according to the editor (including colgroup). **Colspan is not supported.** On even or odd rows or columns, the CSS class `even` or `odd` is placed. The `<th>` tag is used for header columns, if specified. **This part does not support `<thead>` and `<tbody>` tags.**

TIP It is possible to use an iterator to generate dynamic tables.

Listing 36. Example usage of the table part.

```
<div class="element table" data-bsi-element="simple-table" data-bsi-element-  
part="table">  
  <table>  
    <tr>  
      <th>Firstname</th>  
      <th>Lastname</th>  
      <th>Age</th>  
    </tr>  
    <tr>  
      <td>John</td>  
      <td>Doe</td>  
      <td>42</td>  
    </tr>  
  </table>  
</div>
```

3.3.2.8. iterator

An arbitrary element. The attribute `data-bsi-iterator` can be defined on this element. The attribute `data-bsi-iterator-item` must be defined on the element or on one of the included elements. This defines the element to be duplicated. If `data-bsi-iterator` is defined and the iterator does not provide any data, the entire element is removed. Otherwise, only the element to be duplicated is marked with `data-bsi-iterator-item`.

A text field is used as the editor. The name of the iterator can be placed in this text field.

Listing 37. Example usage of the iterator part.

```
<div class="element iterator" data-bsi-element-part="iterator" data-bsi-  
iterator="Customer">  
  <div class="customer" data-bsi-iterator-item="Customer">  
    <dl>  
      <dt>Name</dt>  
      <dd>{Customer.name}</dd>  
      <dt>Address</dt>  
      <dd>{Customer.address}</dd>  
    </dl>  
  </div>  
</div>
```

3.3.2.9. news-snippets

An arbitrary element. Displays a news item. Within the element, elements with the `data-bsi-property` attribute can be defined. This attribute may have the following values: `image`, `title`, `headline`, or `text`. The content of these elements is then replaced accordingly with the value of the selected news item. In that context, elements from property `title`, `headline`, and `text` are mandatory.

If a link in CX is configured into a news snippet, `title` and `image` are automatically supplemented with a link. When generating the news list, the following CSS classes are automatically set on a news snippet to permit better styling of the individual elements:

news-nth-X	the position in the list. E.g., <code>news-nth-9</code>
odd	odd position in the list
even	even position in the list
first	first news snippet
last	last news snippet

Listing 38. Example usage of the news-snippets part.

```
<div class="element news" data-bsi-element="news-snippets" data-bsi-element-  
part="news-snippets">  
  <div class="news-image element">  
      
  </div>  
  <div class="news-text element">  
    <h2 class="ns-title" data-bsi-property="title">Breaking News</h2>  
    <p class="ns-headline" data-bsi-property="headline">Lorem ipsum dolor sit  
amet</p>  
    <p class="ns-text" data-bsi-property="text">Ea vis odio voluptua, pri ei facer  
denique.</p>  
  </div>  
</div>
```

3.3.2.10. form

A `<form>` element. The element must contain a submit button and may also contain a reset button. Those buttons must be `<input>` elements. In the form configuration, the reset button can be hidden or displayed. In that case, the DOM attribute `disabled` is set on the input element.

The form element must contain an element with the `data-bsi-form-validation` attribute, and this attribute must contain an element with the `data-bsi-form-validation-item` attribute.

Within the element with `data-bsi-form-validation`, the element with `data-bsi-form-validation-item` is duplicated for each error in the server-side form validation, and the validation message is inserted as the content of the validation item element.

IMPORTANT

Boolean attributes like `data-bsi-form-validation` require an empty value `=""` or sample value like `="true"`.

In addition to the display in the above-described element, a `data-bsi-form-validation-message` attribute is set for each defective form field. It contains the validation error for the respective field.

The attribute can be used to style the field with CSS if there is an error (CSS selector checks for the presence of the attribute) or to display the error message with CSS and/or JavaScript in the respective field.

The following form fields can be marked as mandatory fields in the content editor. If this is done, the `required` attribute is set in the relevant HTML element. Thus, the mandatory elements are styled using the respective CSS selectors.

To exclude a certain form from CX you set the `data-bsi-form-ignore` attribute on the form tag. Any form with this marker will be left untouched by CX.

Listing 39. Example usage of the form part.

```
<div class="element form" data-bsi-element="slim-form">
  <form class="form form-wrapper" data-bsi-element-part="form" id="form">
    <ul class="form-validation" data-bsi-form-validation="">
      <li class="form-validation-item" data-bsi-form-validation-item=""></li>
    </ul>
    <div class="form-content" data-bsi-dropzone="form-content" data-bsi-dropzone-
allowed-elements="form-field">
      </div>
      <div class="form-button-bar">
        <input type="submit" class="form-button send" value="Senden" />
        <input type="reset" class="form-button reset" value="Zurücksetzen" />
      </div>
    </form>
  </div>
```

3.3.2.11. form-field

A `<div>` element. Contains a `<label>` and an `<input>` element. In the configuration, the label, initial value, mandatory field, and input type can be defined. The following input types are supported:

text	For a simple text field.
email	Input field, that accepts a valid e-mail address.
tel	Input field, that accepts a phone number.
password	A password input field, that shows * instead of the real characters.
number	An input field, that accepts only numbers.
range	An input field, that shows a numeric range slider.
date	An input field, that can be used to select a certain date.

datetime-local	Input field, that accepts a date and a specific time.
time	Input field, that accepts a time.
file	Input field, that can be used to upload a file.

To fixate the form field type, the `data-bsi-form-field-fixed-type` attribute can be used on the element. Doing so allows the setup of a customized date field, for example. The type of the form field is then set and cannot be changed in the content editor.

The attribute `data-bsi-hide-required` controls whether the *mandatory* checkbox is visible and editable in the content editor. If the checkbox is not shown, the original state of the `required` attribute is preserved, as defined by the content element.

IMPORTANT	The attribute <code>data-bsi-hide-required</code> is available from CX 1.3.40 onward
WARNING	Be aware, that not all browsers support all types of input fields. Some browsers require a polyfill to handle certain types of form fields. It is up to the developer to use polyfills where this is required.

Listing 40. Example usage of the form-field part.

```
<div class="form-field" data-bsi-element="date-field" data-bsi-element-part="form-
field" data-bsi-form-field-fixed-type="">
  <label for="date" class="label">Date</label>
  <input id="date" class="input" type="date" value="2009-11-13" required="" />
</div>
```

IMPORTANT	Boolean attributes like <code>data-bsi-form-field-fixed-type</code> require an empty value <code>=""</code> or sample value like <code>"true"</code> .
------------------	--

3.3.2.12. form-checkbox

A `<div>` element. Contains a `<label>` and an `<input>` element with the `checkbox` type.

The attribute `data-bsi-hide-required` controls whether the *mandatory* checkbox is visible and editable in the content editor. If the checkbox is not shown, the original state of the `required` attribute is preserved, as defined by the content element.

IMPORTANT	The attribute <code>data-bsi-hide-required</code> is available from CX 1.3.40 onward
------------------	--

Listing 41. Example usage of the form-checkbox part.

```
<div data-bsi-element="form-checkbox" data-bsi-element-part="form-checkbox"
class="element checkbox">
  <input type="checkbox" id="checkbox1" />
  <label for="checkbox1">Checkbox 1</label>
</div>
```

3.3.2.13. form-textarea

A `<div>` element. Contains a `<label>` and a `<textarea>` element.

The attribute `data-bsi-hide-required` controls whether the *mandatory* checkbox is visible and editable in the content editor. If the checkbox is not shown, the original state of the `required` attribute is preserved, as defined by the content element.

IMPORTANT	The attribute <code>data-bsi-hide-required</code> is available from CX 1.3.40 onward
------------------	--

Listing 42. Example usage of the form-textarea part.

```
<div data-bsi-element="form-textarea" data-bsi-element-part="form-textarea"
class="form-field form-element">
  <label for="textarea" class="label">Description</label>
  <textarea id="textarea" rows="5" cols="20" class="textarea" maxlength="500">Lorem
ipsum</textarea>
</div>
```

3.3.2.14. form-select

A `<div>` element. Contains a `<label>` and a `<select>` element with the `elements` option. The pre-allocation of the element can be controlled with the `data-bsi-value` and `data-bsi-value-list` attributes. The `data-bsi-value` attribute contains the preal-location, and the values available for selection are defined in the `data-bsi-value-list` attribute. A line break (`\n` or `\r\n`) separates each of those.

To fixate the values that are available for selection, the `data-bsi-form-select-fixed-value-list` attribute can be set on the element. Then, the list of values cannot be changed in the content editor. Only the preselection can be changed.

The attribute `data-bsi-hide-required` controls whether the *mandatory* checkbox is visible and editable in the content editor. If the checkbox is not shown, the original state of the `required` attribute is preserved, as defined by the content element.

IMPORTANT	The attribute <code>data-bsi-hide-required</code> is available from CX 1.3.40 onward
------------------	--

WARNING	The <code>multiple</code> attribute is not supported.
----------------	---

Listing 43. Example usage of the form-select part.

```
<div data-bsi-element="form-select" data-bsi-element-part="form-select" class="form-select form-element" data-bsi-value-list="Yes No Unsure" data-bsi-value="No">
  <label for="select" class="label">Decision</label>
  <div class="select-wrapper">
    <select size="1" id="select">
      <option>Yes</option>
      <option selected="">No</option>
      <option>Unsure</option>
    </select>
    <div class="dropdown" role="presentation">a</div>
  </div>
</div>
```

3.3.2.15. form-radio

A `<div>` element. Contains a `<label>` and a `<div>` element as a radio group. The radio group is marked with the `data-bsi-radio-group` attribute. The radio group contains radio items. These are identified with the `data-bsi-radio-item` attribute and contain a `<label>` and an `<input>` element with the radio type. The preselection of this element can be controlled with the `data-bsi-value` and `data-bsi-value-list` attributes. Same as the `form-select` part. The same applies to the `data-bsi-form-select-fixed-value-list` attribute.

The attribute `data-bsi-hide-required` controls whether the *mandatory* checkbox is visible and editable in the content editor. If the checkbox is not shown, the original state of the `required` attribute is preserved, as defined by the content element.

IMPORTANT	The attribute <code>data-bsi-hide-required</code> is available from CX 1.3.40 onward
------------------	--

Listing 44. Example usage of the form-radio part.

```
<div data-bsi-element="form-radio" data-bsi-element-part="form-radio" class="form-radio form-element">
  <label class="label">To be or not to be?</label>
  <div data-bsi-radio-group=""> ①
    <div class="form-radio-item" data-bsi-radio-item=""> ②
      <input type="radio" id="radio1" />
      <label for="radio1">Radio-Button 1</label>
    </div>
  </div>
</div>
```

① The `data-bsi-radio-group` element contains the prototype element.

② The `data-bsi-radio-item` element is the prototype element.

WARNING

Be aware, that `data-bsi-radio-item` must be right inside `data-bsi-radio-group`. Any intermediate element will be removed.

IMPORTANT

Boolean attributes like `data-bsi-radio-item` require an empty value `=""` or sample value like `"true"`.

3.3.2.16. link

The part is defined on the `<a>` element. As an option, the link content can be set with the `data-bsi-link-fixed-inner-html` attribute. If this attribute is present, only the `href` attribute of the link can be edited in the content editor. If the link text is to be placed in a tag within the hyperlink (e.g. in a span next to an `` tag), the optional `data-bsi-link-text-part` attribute can be set on the tag.

Currently it is not possible to put a dropzone inside this link element.

IMPORTANT

Boolean attributes like `data-bsi-link-fixed-inner-html` could require an empty value `=""` or sample value like `"true"`.

Listing 45. Example usage of the link part.

```
<div data-bsi-element="cta" class="element cta">
  <a href="https://www.bsi-software.com/" data-bsi-element-part="link">More</a>
</div>
```

Listing 46. Example usage of the link part with `data-bsi-link-fixed-inner-html`.

```
<div data-bsi-element="cta" class="element cta">
  <a href="[dynamic_link]" data-bsi-element-part="link" data-bsi-link-fixed-inner-
html>
    <span>
      <p> Some static content </p>
      <p data-bsi-element-part="formatted-text"> Add your text here! </p>
    </span>
  </a>
</div>
```

Listing 47. Example usage of the link part with `data-bsi-link-text-part`.

```
<div data-bsi-element="cta">
  <a data-bsi-element-part="link" href="#" role="button">
    <i class="icon icon-028-arrow-back"></i>
    <span data-bsi-link-text-part>The link text is </span>
  </a>
</div>
```

3.3.2.17. social-follow

A `<div>` element, contains `<div>` elements with a `data-bsi-social-follow` attribute. These attributes contain the value of the respective social media service, such as "Facebook", "Twitter", "Pinterest", etc.

The elements with `data-bsi-social-follow` must have the `social-media-item` class. This class is used to manage the visibility of each individual social media channel in the editor.

The elements with `data-bsi-social-follow` must also contain an `<a>` element. The link `href` can be preselected with a URL or with CX links in square brackets (e.g., [Facebook]).

The `<a>` element, in turn, can contain additional elements.

Listing 48. Example usage of the social-follow part.

```
<div data-bsi-element="social-follow" data-bsi-element-part="social-follow">
  <div data-bsi-social-follow="Facebook" class="social-media-item">
    <a href="https://www.facebook.com/">
      
    </a>
  </div>
  <div data-bsi-social-follow="Twitter" class="social-media-item">
    <a href="https://www.twitter.com/">
      
    </a>
  </div>
</div>
```

3.3.2.18. social-share

A `<div>` element, contains `<div>` elements with `data-bsi-social-share`. These attributes contain the value of the respective social media service, such as Facebook, Twitter, Pinterest, or even e-mail.

The elements with `data-bsi-social-share` must have the `social-media-item` class. This class is used to control the visibility of each individual social media channel in the editor.

The elements with `data-bsi-social-share` must also contain an `<a>` element. The link `href` can be preselected with a URL or with Studio links in square brackets (e.g., [Facebook]).

To share it via e-mail, the content of the `href` is then a `mailto:?subject=(subject.with.title)&body=(description.with.url)`

The `<a>` element, in turn, can contain additional elements.

The URL supports a list of wildcards that are specific to the social share part. When they are issued, all wildcards are automatically URL-encoded. These wildcards cannot be selected in the step configuration in Studio. The following wildcards exist:

url	The URL/internet address of the current Studio webpage or the user-defined URL.
description	The description text of the current Studio webpage or the user-defined description text (a maximum of 300 characters).
description.short	Same as description, but a maximum of 140 characters; all additional characters are truncated.
description.with.url	Same as description; in addition, the url is added to the end of the description text.
title	The title of the current Studio webpage or the user-defined title (a maximum of 100 characters).
title.with.url	Same as title, and, in addition, the url is added to the end of the description text.
subject.with.title	Same as title, and, in addition, a text "Share web-site:" is added to the beginning of the title; is used as the subject in an e-mail.
preview.image.url	The URL/internet address for the preview image of the webpage according to the configuration in the content editor.


```

<div data-bsi-element="social-share" data-bsi-element-part="social-share">
  <div class="social-media-info">Share this page</div>
  <div data-bsi-social-share="E-Mail" class="sm-email social-media-item share">
    <a href=
"mailto:?subject=(subject.with.title)&body=(description.with.url)"></a>
  </div>
  <div data-bsi-social-share="Facebook" class="sm-facebook social-media-item share">
    <a href="https://www.facebook.com/sharer/sharer.php?u=(url)" target=
"_blank"></a>
  </div>
  <div data-bsi-social-share="LinkedIn" class="sm-linkedin social-media-item share">
    <a
href="https://www.linkedin.com/shareArticle?mini=true&url=(url)&summary=(title)&source
=FancyUnicorns" target="_blank"></a>
  </div>
  <div data-bsi-social-share="Pinterest" class="sm-pinterest social-media-item
share">
    <a
href="http://pinterest.com/pin/create/button/?url=(url)&description=(title)&media=(pre
view.image.url)" target="_blank"></a>
  </div>
  <div data-bsi-social-share="Twitter" class="sm-twitter social-media-item share">
    <a href="https://twitter.com/intent/tweet?url=(url)&text=(description.short)"
target="_blank"></a>
  </div>
  <div data-bsi-social-share="WhatsApp" class="sm-whatsapp social-media-item share">
    <a href="whatsapp://send?text=(title.with.url)" target="_blank"></a>
  </div>
  <div data-bsi-social-share="Xing" class="sm-xing social-media-item share">
    <a href="https://www.xing.com/spi/shares/new?url=(url)" target="_blank"></a>
  </div>
</div>

```

3.3.2.19. url-provider

IMPORTANT

CX 22.0

The URL provider content element part can be used to load dynamic data inside a story. Inside the content editor, the URL provider part only offers a text field to define its name. When using the content inside your story, each URL provider part will produce a separate outlet. This outlets can be connected to any URL provider (e.g. the *Charts Step*).

Listing 49. Example usage of the URL provider part.

```
<div data-bsi-element="pie-chart" data-bsi-element-part="url-provider" class="mb-4
chart-js">
  
</div>
```

When a story renders the content, the HTML tag with the `url-provider` part will have a `data-bsi-url` attribute. This attribute will contain an absolute URL to the CX server, where the data can be requested.

Using the Chart URL Provider

We created a small Java Script library to use the chart URL provider in a template. Take a look at the repository on [GitHub](#) for more information. To integrate the library, install it with NPM:

```
npm install --save github:bsi-software/bsi-cx-chart#semver:^1.1.2
```

You also must install a chart library of your choice, currently there are only bindings for [Chart.js](#). You can install it using npm:

```
npm install --save chart.js
```

Now, create your content element:

```
<div data-bsi-element="pie-chart" data-bsi-element-part="url-provider" class="chart-
js">
   ①
</div>
```

- ① This will be replaced when the library is initialized, and the data from the URL provider is available. So it's recommended to put some placeholders here.

To use the library, you must initialize it:

```

import 'chart.js/auto'; ①

import {ChartConfig, ChartUrlProvider} from '@bsi-cx/chart';
import ChartConfigColor from '@bsi-cx/chart/src/config/color';

/**
 * @type {{border: ChartConfigColor, background: ChartConfigColor}[]}
 */
const colors = [
  ChartConfigColor.of('#ff6384ff', '#ff638466'),
  ChartConfigColor.of('#36a2ebff', '#36a2eb66'),
  ChartConfigColor.of('#cc65feff', '#cc65fe66'),
  ChartConfigColor.of('#ffce56ff', '#ffce5666')
]; ②

const config = new ChartConfig()
  .withColors(...colors);

document.querySelectorAll('.chart-js') ③
  .map(element => new ChartUrlProvider(element, config))
  .forEach(chart => chart.render());

```

- ① Initialize the Chart.js library.
- ② Optional, can be used to define your own set of colors.
- ③ Select all URL provider parts you want to initialize with the `ChartUrlProvider`.

3.4. Data BSI Attributes

Data BSI attribute are regular DOM element attributes, starting with `data-bsi-`. In a CX design they must be used to amend CX specific semantic information to the DOM or to control processing logic while the DOM is rendered by the CX server.

3.4.1. Control Attributes

The presence of an HTML element in the generated content can be controlled by using `data-bsi-remove-if`.

- **draft**: The element is only inserted if the content is being edited in the editor.
- **live**: The element is only inserted if the content is driven by a story.
- **authenticated**: The element is only inserted if an authenticated visitor exists on the CX HTTP session.

The attribute value may have a negate operator (!). Using this operator you can negate the condition, for instance `!authenticated` will only be removed in case the visitor is *not* logged in.

A common and recommended use of `data-bsi-remove-if="draft"` is to suppress executing JavaScript in the content editor, which may lead to unintended side effects .hiding information in draft mode.

```
<script data-bsi-remove-if="draft">
// this script will not mess around with your content while being edited in the
content editor
</script>
```

A common use case of `data-bsi-remove-if="live"` mode is to display context information to the user that should only be present while editing a content. .hiding information in live mode.

```
<p data-bsi-remove-if="live">This hint will never be shown in any content driven by a
story.</p>
```

WARNING

`data-bsi-remove-if` must not be used for tag elements containing content elements! Otherwise, those cannot be edited in draft mode.

data-bsi-hide-edit-button may be used to prevent editing of a content element. The attribute must be located on the same tag as the `data-bsi-element` attribute.

Listing 50. hiding the content element edit button

```
<div data-bsi-element="..." data-bsi-hide-edit-button="true">...</div>
```

data-bsi-hide-move-button, **data-bsi-hide-copy-button**, and **data-bsi-hide-remove-button** are working in the same manner, but on dropzone level. They will prevent from moving, copying, or removing content elements on a dropzone.

Listing 51. prevent moving, copying and removal of content elements.

```
<div data-bsi-dropzone="..." data-bsi-hide-move-button="true" data-bsi-hide-copy-
button="true" data-bsi-hide-remove-button="true">...</div>
```

3.4.2. Form Processor Attributes

The CX server may implement programmed backend logic to validate and process a specific form in a CX website. The following attributes control which form processor is called on a form submit. A CX project that implements its own backend logic may contribute its own form processors.

Set the `data-bsi-form-type` on a form element to specify which form processor handles the form on submit. Predefined values are `login` and `change-password`. A CX project may define custom values, which must map to a form processor implementation in the CX backend server.

The `data-bsi-show-form-type-field` controls whether the CX content editor shows a drop-down menu to choose a form type in the Form element part. This gives a design author the freedom to create a generic content element for forms, where the content author can choose to create a standard form or a form using a form processor via drop-down menu, or to provide a complete form as a dedicated form element, where the content author cannot change the form type in the content editor.

Set the `data-bsi-form-type` attribute on a form field element to mark a field interpreted by a form processor. For instance a `login` form requires the form to have two fields, one must define `data-bsi-form-type="user-id"` and the other `data-bsi-form-type="password"` in order to pass the form processor validation. Each form process may define custom values for this attribute.

3.5. Websites

3.5.1. Terms

- Website: The complete website. Contains several pages in different languages.
- Page: A single page within a website. Pages appear in the navigation and have a content in different languages.
- Include: A content snippet, which is saved locally once and can be referenced at any point within the website. We differentiate between built-in includes and custom includes. Built-in includes are content snippets that are included in the design.zip, custom includes are created by the user based on a content-element.

3.5.2. Concepts

- CX-Placeholders can be used on every page of a website. Their scope generally involves a single page. For example the placeholder `"{Subject}"` has to be mapped once for each page that uses it. A placeholder within an include (e.g. header or footer) only has to be mapped once. An example for that case would be the display of `"{Username}"` within the header of a portal-website. Placeholders can *not* be used within page-titles or the navigation.
- Custom includes *can't* be nested. Technically, that would be possible, but it would need many checks and visualizations inside the content editor to detect potential infinite rekursions and give feedback to the user through reasonable error messages. For that reason we currently don't support that feature.
- An include can only be used *once* within the same page.
- Multilingualism: All contents and their includes are always created in every active language for their website. E.g.: The website is configured to be German and Italian. On the creation of a new page, two pages will be created: one for Italian and one for German (including their dummy-contents). Within the UI there is no option to create a single page in one specific language only. If necessary, a CX-condition may be used to hide a page for a specific language inside the navigation-tree.

3.5.2.1. Handlebars, Templates

The template-engine *Handlebars* is available to the design developer on runtime. The templates are parsed server-sided. We're using the [Java-Port by JKnack](#). In the design, handlebars templates are important especially for display of the navigation.

Handlebars templates can only be used for parts of a website that *can't* be edited by the user. We set that rule, because templates are parsed server-sided. When the content-editor receives the HTML source, the templates have already been resolved. If the user would edit a content-element that contained handlebars syntax, we could only save the rendered content that the user edited.

The original handlebars code would be lost.

For that reason, handlebars should mostly be used for includes that *can't* be edited. Examples for such includes are the master ([design.html](#)), the navigation or areas outside a dropzone.

This mechanism can also be used conciously to set multilingual default texts, which are deposited inside the [website.json](#) file. An example for an include for the website header in handlebars code could look something like this:

```
<div data-bsi-element="master-header">
  
  <h1 data-bsi-element-part="plain-text">{{bsi.nls "website.title"}}</h1>
</div>
```

On the initial rendering of the page inside the content editor, the text "[website.title](#)" is resolved. As the user saves the page, the initial handlebars code is substituted by the actual text.

Handlebars syntax may also be used for the [content-elements.html](#) file; especially for the NLS-helper to define multilingual default texts for each content-element.

Important: Files that contain handlebars syntax *aren't* valid HTML documents. They shouldn't be handled with HTML parsers like JSoup, otherwise the handlebars syntax gets corrupted. Files that include handlebars code should use the suffix *.hbs*. *IntelliJ* supports this syntax with respective plugins.

3.5.2.2. Mustache Helpers

Besides the built-in standard helper, BSI CX may also use the following helpers:

3.5.2.3. bsi.nls

This helper is used to display localized texts or resources inside the design. The syntax looks like this: `{{bsi.nls "[textKey]"}}`.

It can also be used inside the [content-elements.html](#) file to deliver localized default texts for different languages.

Text keys are created inside the [design.json](#) at the property [nls](#). Text that doesn't depend on the locale is defined with the key ***. The resolution follows from the most specific locale up to the default, e.g.:

```

{
  "nls": {
    "hello.world": {
      "*": "Hello World",
      "de": "Hallo Welt",
      "de-CH": "Sali Wält"
    }
  }
}

```

3.5.2.4. Website variables

Following variables are available to the design developer on runtime. The variables can be used in mustache-templates, e.g.: `{{navigation.activePage.contentId}}`.

3.5.2.5. General (Root)

- **title** (String): Title of the website. Matches the field "Title" in the website-masters editor. Can be used in combination with the title of the current page to display the title of the HTML document, e.g.: `{{title}} | {{navigation.activePage.pageTitle}}`.
- **language** (String): Current language as it's used in the `lang` attribute of the HTML tag, e.g.: `de`.
- **locale** (String): Complete locale. The locale is used to resolve text keys through the `bsi.nls` helper, e.g. `de-CH`.
- **navigation** (Object): Contains the hierarchical navigation tree, example follows.
- **paginationInfo** (Object): Contains the information for pagination. This object is only filled if the pagination feature is specified in the `design.json` file.

3.5.2.6. Navigation Item

- **active** (Boolean): True, when the navigation element (page) or the HTML site is active.
- **id** (UUID): Primary key according to table `BSI_WEBSITE_REV_NAV#ID`.
- **contentId** (UUID): Primary key according to table `BSI_WEBSITE_REV_CONT#ID`.
- **url** (String): URL according to table `BSI_WEBSITE_REV_NAV_LOCALE#URL`.
- **title** (String): Title according to table `BSI_WEBSITE_REV_NAV_LOCALE#NAVIGATION_ITEM_TITLE`.
- **folder** (Boolean): True, when `BSI_WEBSITE_REV_NAV_LOCALE#NAV_ITEM_TYPE = "folder"`. Used for separation between pages and folders.
- **level** (Integer): Recursion-level of the navigation element, starting with 0 = Root.
- **image**: Absolute URL for the navigation image according to table `BSI_WEBSITE_NAVIGATION#IMAGE`.
- **metadata**: JSON metadata object according to table `BSI_WEBSITE_NAVIGATION#METADATA`.
- **localizedUrls**: Object with URL as value for the language-change widget and locale as key.

3.5.2.7. Pagination Info

- **currentPageNo**: Current page number
- **numDataRecordsPerPage**: Number of records per page
- **firstDataRecordOffset**: Index of the first record on the current page
- **totalNumDataRecords**: Total number of existing records
- **firstPageUrl**: URL of the first page
- **lastPageUrl**: URL of the last page
- **lastPageNo**: Page number of the last page
- **previousPages**: Contains information on lower page numbers that are relevant for pagination.
 - **pageNo**: Page number
 - **url**: URL of the page
 - **index**: Offset of the page number to the current page number
- **nextpages**: Contains information on higher page numbers that are relevant for pagination. The information set is exactly the same as for **previousPages** and is therefore not listed anew here.

3.5.2.8. Example JSON

This example shows the possible content of a variables object on runtime.

```
{
  "language": "de",
  "locale": "de-CH",
  "navigation": {
    "activePage": {
      "id": "1006",
      "contentId": "2006",
      "url": "ueber-bsi.html",
      "title": "Über BSI",
      "level": 0,
      "active": true,
      "localizedUrls": {
        "de-CH": "ueber-bsi.html",
        "en-CH": "about-bsi.html"
      }
    },
    "index": {
      "id": "1001",
      "contentId": "2001",
      "url": "index",
      "title": "Willkommen bei BSI",
      "level": 0
    },
    "items": [
      {
```



```

    "id": "1002",
    "contentId": "2002",
    "url": "loesungen.html",
    "title": "Lösungen",
    "level": 0
  },
  {
    "id": "1003",
    "contentId": "2003",
    "url": "branchen.html",
    "title": "Branchen",
    "folder": true,
    "level": 0,
    "items": [
      {
        "id": "1004",
        "contentId": "2004",
        "url": "banking.html",
        "title": "Banking",
        "level": 1
      },
      {
        "id": "1005",
        "contentId": "2005",
        "url": "health.html",
        "title": "Health",
        "level": 1
      }
    ]
  },
  {
    "id": "1006",
    "contentId": "2006",
    "url": "ueber-bsi.html",
    "title": "Über BSI",
    "level": 0,
    "active": true
  }
]
}

```

```

{{#navigation}}
<nav class="navigation">
  {{#items}}
    <ul class="nav-item-container level-root">
      <li class="nav-item level-root">
        <!-- root level -->
        {{#active}}
          <span class="nav-item-active">{{title}}</span>
        {{/active}}
        {{^active}}
          <a href="{{url}}">{{title}}</a>
        {{/active}}
      </li>
    {{#items}}
    <ul class="nav-item-container level-1st">
      <!-- 1st level -->
      <li class="nav-item level-1st">
        {{#active}}
          <span class="nav-item-active">{{title}}</span>
        {{/active}}
        {{^active}}
          <a href="{{url}}">{{title}}</a>
        {{/active}}
      </li>
    </ul>
  {{/items}}
</ul>
{{/items}}
</nav>
{{/navigation}}

```

3.5.3. Configuration files

This chapter describes the necessary configuration files.

3.5.3.1. Website-specific metadata inside design.json

The *design.json* file is a metadata file for structured configuration data.

Example for the design.json file for website designs:

```

{
  "schemaVersion": "1.0",
  "defaultLocale": "de",
  "locales": ["de", "en"],
  "nls": {
    "page": {
      "*": "Seite",
      "en": "Page"
    }
  },
  "website": {
    "maxNavigationLevel": 2,
    "pagination": {
      "numDataRecordsPerPage": 20,
      "numAdjacentPages": 3
    },
    "includes": {
      "__page__": {
        "name": "Vorlage für Inhaltsseiten",
        "reference": "__page__",
        "file": "include/page.html"
      },
      "navigation": {
        "name": "Navigation",
        "reference": "navigation",
        "file": "include/navigation.hbs"
      },
      "header": {
        "name": "Kopfzeile",
        "reference": "header",
        "file": "include/header.html",
        "editable": true
      },
      "pagination-element": {
        "name": "Pagination",
        "contentType": "pre-defined",
        "file": "include/pagination-element.hbs",
        "editable": false
      },
      "footer": {
        "name": "Fusszeile",
        "reference": "footer",
        "file": "include/footer.html",
        "editable": true
      }
    }
  }
}

```

- **schemaVersion**: Needs to be set to 1.0, might be relevant for changes in later versions.

- **defaultLocale**: Default language (must be contained in **locales**).
- **locales**: List with languages (e.g.: **en**, **de**, **de-CH**).
- **nls**: Contains a map with NLS keys, which can be used within the .hbs files with **bsi.nls**.
- **website**: **maxNavigationLevel** defines the maximum depth of the navigation (level of the hierarchy) that can be displayed by the design. To allow better structuring of pages and folders, the website editor allows the creation of websites with greater depth, but such pages or folders can't be selected through the navigation on runtime. However, an internal link from one page to another page with greater depth is still possible.
- **website**: **pagination** is used for pagination (see [Pagination](#)). The property **numDataRecordsPerPage** can be used to specify how many records are to be displayed simultaneously on a page. The property **numAdjacentPages** describes how many lower and higher page numbers are to be displayed in the pagination navigation. For example, if **numAdjacentPages** is set to 3 and you are on page 2, then the maximum number of pages displayed in the navigation is 1, 2, 3, 4 and 5.
- **includes**: See below.

3.5.4. Includes

All files that are inherited to the database on the creation of a website as navigation item or include have to be referenced in the **design.json** file. All includes from the design.zip are inherited automatically with **content-type=built-in**, unless **content-type=pre-defined** is set explicitly.

The file **design.html** matches the master and is automatically created inside the website internally as an include with the reference **master**. For that reason this include should *not* be defined in the design.json file, as it is defined already.

The minimal include is **page**, which works as a template for new pages within a website.

The key, which is used for the **bsi:include**-tags, is used as a unique reference to the include.

For each include, the following properties can be configured:

- **name**: Displayed name of the include.
- **file**: Points to a file inside the zip archive, relative to the root of the zip file. Only .html and .hbs are allowed.
- **contentType**: **built-in** (default, if not set manually), **pre-defined** and **user-defined**.
- **editable**: True, when the include can be edited in the content editor by the user. Default is true. Must be **editable: false** for all includes that use handlebars.

3.5.4.1. design.html

The **design.html** file is equivalent to the "master" of the website. On the creation of a website, an include "Design" with this file's content is created. This include *can't* be edited by the user. However, the design can reference other includes that may be edited, so that e.g. the footer can be edited by the user.

A design can contain exactly one include for a page (attribute "pageid"). For the editor this include

has a special meaning: when displayed in the editor, the full page (including the master) is displayed, but when saved, only the content of the respective page is saved.

Advice: There should be no tags of the template-engine used for includes or pages that the user may edit.

```

<html>
<head>
  <title>{{site.title}} | {{navigation.activePage.title}}</title>
</head>
<body>

<header>
  <h1>Unicorn Inc.</h1>
  {{#navigation.index.active}}
    
  {{/navigation.index.active}}
  {{^navigation.index.active}}
    <a href="{{navigation.index.url}}"></a>
  {{/navigation.index.active}}
  <bsi:include id="navigation" editable="false">
</header>

  <!--
  ActivePage is always set by the framework. When the root-URL '/' is loaded,
  'index' is used automatically.
  When the include-tag is resolved, it's replaced by the respective HTML-snippet.
  Having a single root-element (e.g.: <div>, <p>, ...), that capsulates the entire
  DOM of the include, is a technical requirement for any include.
  When changed, the attribute "data-ce-include-id" is set on this element with the
  value of the original attribute "id".
  -->
  {{#response.ok}}
    <bsi:include reference="{{navigation.activePage.contentId}}" editable="true">
  {{/response.ok}}
  <!--
  This include is only loaded when the reposnse is != OK. It takes a special
  function to set response.ok to false in the content-editor, to even make it editable
  in the editor.
  -->
  {{^response.ok}}
    <bsi:include id="error" editable="false">
  {{/response.ok}}

</body>
</html>

```

The include "footer-text" loads the built-in include footer-text.html. This file looks like this:

```
<div data-ce-content-element="footer-text"
      data-ce-element-part="formatted-text">
  This is a default-text for the footer. This text can be changed in the editor by the
  user.
</div>
```

3.5.5. Content-Editor

Before the content is displayed in the editor, it has to be preprocessed by the `WebsiteProcessor`. It resolves includes and starts the template-engine. Just like the live-website, the content editor also creates an object with website variables.

Here is an example for content, when it's loaded from the database:

```
<bsi:include reference="footer-text">
  {{title}}
```

Here is an example for content, when it's shown in the editor:

```
<div
  data-bsi-include-reference="footer-text"
  data-bsi-element="footer-text"
  data-bsi-element-part="formatted-text">
  Ipsum Lorum
</div>
BSI Website
```

3.5.6. Pagination

Pagination can be used to distribute data supplied by an external system (e.g. via REST) to several pages on a website, so that only part of the data is displayed on one page. With *Next* and *Back* buttons on the website page, it is then possible to display a separate set of data on each page.

The pagination must be defined via `website: pagination` in `design.json` and provided via an include. The include must exist in the `design.json` file and have the special content type `pre-defined` (see [Website-specific metadata inside design.json](#)). The include itself should have the extension `.hbs` and use handlebars. The possible values for the Mustache templates are documented in the chapter on website variables (see [Pagination Info](#)). Besides, the attribute `data-bsi-hide-edit-button` should be set, as includes that use handlebars should not be editable in the website editor. The following is an example of a possible template:

```

<div class="pagination-element" data-bsi-element="pagination-element" data-bsi-hide-
edit-button="true">
  <a href="{{pagination.firstPageUrl}}">First Page</a>
  {{#each pagination.previousPages}}
    <a href="{{url}}">[{{pageNo}}]</a>
  {{/each}}
  {{pagination.currentPageNo}}
  {{#each pagination.nextPages}}
    <a href="{{url}}">[{{pageNo}}]</a>
  {{/each}}
  <a href="{{pagination.lastPageUrl}}">Last Page</a>
</div>

```

This template (include) is then rendered by the template engine at runtime. One possible output is the following. Everything except page 20 are links pointing to the respective pages.

[First Page] [18] [19] 20 [21] [22] [Last Page]

The data to be rendered via pagination can finally be displayed with the table content element. For this, a content element with the element part `table` should be extended by the attribute `data-bsi-show-pagination-field`. The following is an example of a possible table template:

```

<div data-bsi-element="table" data-bsi-element-part="table" data-bsi-show-pagination-
field="">
  <table>
    <tr>
      <th>#</th>
      <th>Title 1</th>
      <th>Title 2</th>
      <th>Title 3</th>
    </tr>
    <tr>
      <td>1</td>
      <td>Text 1</td>
      <td>Text 2</td>
      <td>Text 3</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Text 1</td>
      <td>Text 2</td>
      <td>Text 3</td>
    </tr>
  </table>
</div>

```

If the attribute `data-bsi-show-pagination-field` is set, the checkbox with `Pagination` will be displayed in the configuration of the table content element. In order to use pagination, the

checkbox should be activated.

Inhalt: Table

Table

#	Title 1	Title 2
1	Text 1	Text 2

☒ with header

☒ Dynamic table

☒ with Pagination ⓘ

In dynamic tables, the (only) row will be repeated for each entry of an entity. Placeholders can be defined in this format:
{Entity.Attribute} Example:
{Customer.Name},
{Customer.Age}

OK

Cancel

☐ Save as include

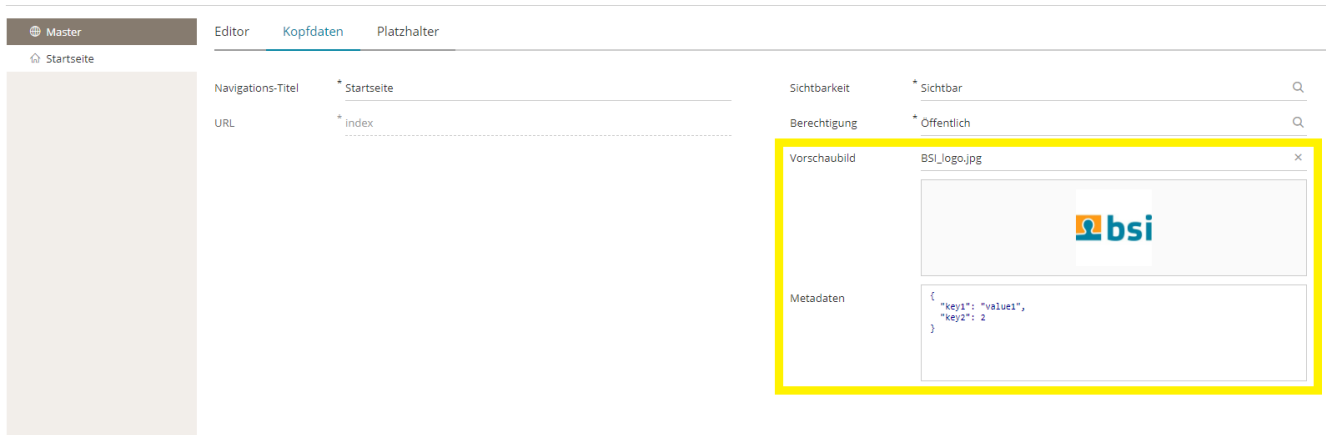
☐ Variant

3.5.6.1. Limitations

There are some limitations to pagination:

- Pagination has only been implemented for tables (i.e. the element part **table**).
- The pagination can only be used directly in a website. We do not support the embedding of such a table in an include. Other content (for example for landing pages) is also not supported.
- To enable a meaningful display of the handlebars code in the website editor, the Mustache templates are rendered when the editor is initially loaded. Afterwards, the rendering happens only exceptionally. This can lead to the display being incorrect if page-dependent elements (such as `{{navigation.activePage}}`) are used in the template.

3.5.7. Metadata / Images for the navigation



Each page and each folder of a website can have JSON metadata and an image to describe the navigation. These two elements are special, because they can only be edited programatically and within a project-specific step. A possible use case is a REST webservice, which is embedded to the design through javascript. The project-specific step can return the image and the metadata on a GET request. This allows the rendered website to give images and information to the navigation.

These two attributes are invisible per default. However, they can be enabled thorough the CX setting *Website-Editor*. Both attributes are the same for every language, so it's *not* possible to deposit different images for a German and an English website.

3.5.8. Security - Google reCAPTCHA

It's highly advised to use the Google reCAPTCHA service "I'm not a robot", when using username/password authentication with non-internal users (public). For that we use the optional CX module `com.bsiag.studio.media.googlerecaptcha`, which can be configured in the CX settings.

Example for including the dependency in the CX app and the dev pom.xml:

```
<!-- google reCAPTCHA service for website step -->
<dependency>
  <groupId>com.bsiag.studio</groupId>
  <artifactId>com.bsiag.studio.media.googlerecaptcha</artifactId>
</dependency>
```

A script can be included on the website's design.html:

```
<script src="https://www.google.com/recaptcha/enterprise.js" async defer></script>
```

The captcha-field can be added to the login form of the content-elements.html as follows:

```

<form ... data-bsi-element="login-form">
...
<!-- Content Element: Google reCAPTCHA Enterprise -->
<div data-bsi-element-part="form-field" class="form-field-container form-field">
  <label data-bsi-remove-if="live">{{bsi.nls "captcha"}}</label>
  <input data-bsi-remove-if="live" type="text" data-bsi-form-field-type="captcha"
value="Google reCAPTCHA" id="g-recaptcha-response" name="g-recaptcha-response">
  <div class="g-recaptcha" data-sitekey="the site key from the google api recaptcha
registration" data-action="login">
    </div>
  </div>
</div>

```

The label text can be added to the design.json as follows:

```

"captcha": {
  "*": "Captcha",
  "de": "Captcha"
},

```

After that, the field ID of the captcha field (on the website, that uses the design, simply open the edit dialogue and scroll down to the captcha field ID) has to be set to `g-recaptcha-response`. This value has to match the settings of the Google reCAPTCHA in the administration view.

The last step is to generate a reCAPTCHA key on the Google Cloud of the customer/company and register it in the settings under *Administration - Customer Experience - Settings - Google reCAPTCHA Settings*.

For local testing, the URL of the BSI CX website *can't* be `localhost`. For this purpose, the local domain `bsiag.local` may be used. Simply go to the administration view - *CX - Settings* under *base URL for public links* and register a second path, e.g. `bsiw0160`:

```

{
  "_type": "start.BaseUrlSetting",
  "default": false,
  "inboundBaseUrl": "http://bsiw0160.bsiag.local:8085/inbound/default",
  "qualifier": "default",
  "resourceBaseUrl": "http://bsiw0160.bsiag.local:8085/dev/resources"
}

```

Then select this URL on the smart field `base path` on the website step.

Here is an approximate guide for the creation of a reCAPTCHA site on the example of BSI (customers have to adapt it to their company):

- Create project: BSI CX - Website Step → `projectId=bsi-cx-website-step`
- Add the service `reCAPTCHA Enterprise API` to the project

- Create login/access data for that api by creating a service account **Service account for bsi cx** with role **recaptcha enterprise agent** → `serviceAccountId=service-account-for-bsi-cx`
- Add access key (json file) for that agent service account in IAM section of Google Cloud → `credentials.json` file
- Create reCAPTCHA site key [here](#) (this is just an example)
- Add domains `bsi-software.com` and `bsiag.local` to site → `recaptchaSiteKey=6Lcxp-...._lWrq`

Documentation:

- [Choosing the setup method for reCAPTCHA Enterprise](#)
- [reCAPTCHA Enterprise client libraries](#)
- [Marketplace reCAPTCHA Enterprise API](#)

3.5.9. Creation of websites from existing landingpage templates

The following steps are necessary to create a website template from an existing landingpage template:

1. For navigation inside the website, the navigation has to be created inside an include.
2. Includes for the header- and footer-area have to be defined inside the **design.json** file. Furthermore, the *Page-Include* has to be defined. That area contains the proper content of the website.
3. For landingpages images and other resources of the design can be referenced with relative paths. Because the base URL of a website points to the location of the website and not to the design, the handlebars template `{{designBaseUrl}}` needs to be put in front of every design resource URL. For example, `"img/my-image.jpg"` has to be converted to `"{{designBaseUrl}}/img/my-image.jpg"`. The handlebars template is replaced with the design's correct URL by the CX server on runtime.

4. CX Design Build

WARNING

As of BSI CX 22.0, the [design build](#) has been introduced and supersedes the previous CX design creator. Please refer to version 16.3 of the [CX design documentation](#) for more information about the CX design creator.

The [design build](#) was developed to facilitate the creation of templates. The NPM build compiles ZIP files that are uploaded to BSI CX using a webpack build.

The CX design build has the following functionality and tasks:

- Parsing Twig templates for HTML
- Formatting the generated HTML "nicely" and validating it
- Parsing SCSS/SASS to CSS
- Minifying and autoprefixing CSS
- Merging and minifying JavaScript
- Creating the ZIP file for the upload to BSI CX

A detailed description of the application and the configuration options of the design build are documented in the [design build wiki at GitHub](#).

Please note: When creating designs with the CX design build, the generated artifacts (i.e., HTML, CSS, and JavaScript files) should not be directly edited. Instead, any modifications should be made in the source code itself.

Of course, you can also use a separate tool stack to develop a design. We do not recommend the development of a design directly in HTML, CSS, and JavaScript. This may work with small templates, but it quickly becomes confusing and error prone.

5. Migrate an existing design to the new build

This chapter provides a short migration guide to the conversion of an existing design for CX version 1.3 or lower to the [new design build](#). For basic information on the new design build, please consider the [corresponding Wiki page on GitHub](#).

As a first step of the migration, it is required to set up the project structure. It's recommended that you start with the [Scaffold Design](#) instead of creating the structure all by yourself. Also consider reading [this chapter](#) before you start with the migration.

5.1. Content Elements

Now you can start the migration by transforming the content elements. This step will be explained on an example:

Listing 52. Content element from the old design creator build.

```
<div class="content-element image-wrapper" data-bsi-element="image">
  <div class="row">
    <div class="flex">
      <div class="element">
        <div class="image">
          
        </div>
        <div class="image-legend" data-bsi-element-part="plain-text">{{ legend
?: "Lorem ipsum image description" }}</div>
      </div>
    </div>
  </div>
</div>
```

Listing 53. Corresponding entry in the design.properties file.

```
element.image.label=Image
element.image.description=with description
element.image.icon=image
element.image.parts.image.label=Image
element.image.parts.plain-text.label=Description
```

The content elements are usually organized in Twig files inside a `content-elements` folder. While the corresponding properties are placed in a single `design.properties` file. In the new build process, it's recommended to keep it all together in a single folder and separated from any other content element. So take a look on the existing content elements from the scaffold design. They are all nicely organized in separate folders. So create a new folder for your own element:

Listing 54. Folder structure of a single content element.

```
.
└── image
    ├── index.js ①
    ├── placeholder.jpg ②
    ├── template.twig ③
    └── styles.less ④
```

- ① This file will contain the content element specification.
- ② Feel free to place any content element related assets here.
- ③ The Twig template for our content element.
- ④ Any content element related Stylesheets. This can also be a LESS or SASS file.

Create the `index.js` and `template.twig` files inside this folder and feel free to add any related assets too. In our example, we add the `placeholder.jpg` here. Now we can create the `template.twig`:

Listing 55. The transformed template for the content element.

```
<div class="content-element image-wrapper" data-bsi-element="image">
  <div class="row">
    <div class="flex">
      <div class="element">
        <div class="image">
          
        </div>
        <div class="image-legend" data-bsi-element-part="plain-text">{{ legend
?: "Lorem ipsum image description" }}</div>
      </div>
    </div>
  </div>
</div>
```

Listing 56. The content element specification in the `index.js` file.

```
require('./styles.less'); ①

const {cx, Icon} = require('@bsi-cx/design-build'); ②

module.exports = cx.contentElement
  .withElementId('image')
  .withLabel('Image')
  .withDescription('with description')
  .withIcon(Icon.IMAGE)
  .withFile(require('./template.twig'))
  .withParts(
    cx.part.image
      .withLabel('Image'),
    cx.part.plainText
      .withLabel('Description'));

```

① Include any related stylesheets right at the beginning of your content element specification.

② Be sure you use `require()` instead of `import`.

NOTE

Be aware, that there are some advanced features available with the new design build. For example [referenced content element parts](#), which can be useful in complex content elements.

Once you have finished creating your content element, you can include them in your design specification. The design specification is contained in the `design.js` file and replaces the `design.properties`. You find the `design.js` at the root folder of your template.

Listing 57. The design specification for the transformed design.

```
require('./styles/styles.scss'); ①

const {cx, Design} = require('@bsi-cx/design-build');

/**
 * @type {Design}
 */
module.exports = cx.design
  .withTitle('BSI - Landingpage') ②
  .withAuthor('BSI Business Systems Integration AG') ③
  .withDate('14.04.2022')
  .withContentElementGroups(
    cx.contentElementGroup
      .withGroupId('content')
      .withLabel('Content')
      .withContentElements(
        require('./content-elements/content/image'), ④
        require('./content-elements/content/text')));

```


- ① You can also include global stylesheets (CSS, LESS or SASS) here.
- ② Title of this template, corresponds to the `template.name` from the `design.properties` file.
- ③ The author of this template, corresponds to the `template.author` from the `design.properties` file.
- ④ This is our transformed content element. The path points to the folder. Since it contains an `index.js`, it's not required to mention the filename here.

5.2. Styles

If your content element uses any [Styles](#), you must transform them to a Java Script styles definition:

Listing 58. The style definition in the old `design.properties` format.

```
style.element-width.label=Element width
style.element-width.class.element-with-unset.label=None
style.element-width.class.element-width-full.label=Full
style.element-width.class.element-width-left-right.label=Label left, field right
```

The transformation to the new format is pretty straightforward:

Listing 59. The style definition in the new Java Script format.

```
const {cx} = require('@bsi-cx/design-build');

module.exports = cx.style
  .withIdentifier('element-width')
  .withLabel('Element width')
  .withCssClasses(
    cx.cssClass
      .withLabel('None')
      .withCssClass('element-with-unset'),
    cx.cssClass
      .withLabel('Full')
      .withCssClass('element-width-full'),
    cx.cssClass
      .withLabel('Label left, field right')
      .withCssClass('element-width-left-right'));
```

It's recommended to place the style definition in a separate Java Script file. If your style is exclusively for one element, you can place the file right next to the element definition. Otherwise, you should put it in the `configs/styles` folder inside your template root. To use a style in your content element, simply reference the file:

Listing 60. Use a style definition in your content element.

```
const {cx, Icon} = require('@bsi-cx/design-build');

module.exports = cx.contentElement
  .withElementId('image')
  .withLabel('Image')
  .withDescription('with description')
  .withIcon(Icon.IMAGE)
  .withFile(require('./template.twig'))
  .withStyleConfigs(require('../../../../configs/styles/fade-out')) ①
  .withParts(
    cx.part.image
      .withLabel('Image'),
    cx.part.plainText
      .withLabel('Description'));
```

① Just pass the style definition to the builder using `withStyleConfigs()` and `require()`.

5.3. Design and Preview Templates

To bundle your design and preview templates, create a new `preview.twig` and `design.twig` file inside the template root folder. To prevent code duplication, it's recommended to create a `_layout.twig` template and extend it in your `design.twig` and `preview.twig`. Take a look at the following example:

Listing 61. The contents of a simple `_layout.twig` template.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8"/>
  <title>{{ properties.title }}</title>
  <meta name="viewport" content="width=device-width, initial-scale=1"/>
  <link rel="stylesheet" href="{{ bsi_cx_css_href() }}" /> ①
</head>
<body class="d-flex flex-column min-vh-100">
{% apply spaceless %}
  <header id="header" data-bsi-dropzone="header">
    {% block header %}{% endblock %}
  </header>
  <main id="content" data-bsi-dropzone="content">
    {% block content %}{% endblock %}
  </main>
  <footer id="footer" data-bsi-dropzone="footer">
    {% block footer %}{% endblock %}
  </footer>
{% endapply %}
  <script src="{{ bsi_cx_js_module_runtime_href() }}" data-bsi-remove-if="draft"
defer="defer"></script> ②
  <script src="{{ bsi_cx_js_module_href('main') }}" data-bsi-remove-if="draft"
defer="defer"></script> ③
  {{ bsi_cx_js_module_missing_chunks_import() }} ④
</body>
</html>
```

- ① Use the `bsi_cx_css_href()` Twig helper to include your stylesheets. More options can be found [here](#).
- ② You can skip this line if you don't have any Java Script modules configured.
- ③ Use the `bsi_cx_js_module_href()` Twig helper to include a Java Script module. You can skip this line if you don't have any Java Script modules configured.
- ④ This imports all missing chunks, related to Java Script modules. You can skip this line if you don't have any Java Script modules configured. More information on including Java Script in your design can be found [here](#).

Now you can extend the blocks from the `_layout.twig` in your `design.twig` and `preview.twig`. The following example illustrates this step for the `design.twig`:

Listing 62. The `design.twig` extends the `_layout.twig`.

```
{% extends '_layout.twig' %}

{% block header %}
    {% include './content-elements/header/banner/template.twig' %}
{% endblock %}

{% block content %}
    {% include './content-elements/content/title-h1/template.twig' %}
    {% include './content-elements/content/text/template.twig' %}
{% endblock %}

{% block footer %}
    {% include './content-elements/footer/two-col/template.twig' %}
{% endblock %}
```

5.4. Java Script

You have two options to include your Java Script code:

1. Place them in the `static` folder inside your templates root and include them with `bsi_cx_asset()`.
2. Create a module in the `modules` folder and take full advantage of all modern Java Script.

5.5. Stylesheets

Your global can place your global stylesheets directly in the `styles` folder, located at your template root. You can also place any content element specific styles here instead of moving them to the specific content element's own folder. To bundle your stylesheets, you must reference them in your `design.js`:

Listing 63. Including a stylesheet in your design.

```
require('./styles/styles.scss');
```

5.6. Properties

The new design build uses Java Script files instead of YAML. The `properties.js` file is a CommonJS module, that exports a simple Java Script object `{}`. Feel free to organize your properties as you wish. To access the properties inside your stylesheets and templates, you can use the `bsiProperty()` helper. Take a look at the following example:

Listing 64. Old YAML properties file.

```
styles:
  primary-color: '#ed4546'
  secondary-color: '#efefef'
```

Listing 65. New Java Script properties file.

```
const {css} = require('@bsi-cx/design-build');

module.exports = {
  styles: {
    'primary-color': css.color('#ed4546'),
    'secondary-color': css.color('#efefef')
  }
};
```

TIP

Use the `css` helper to get the most out your properties. More information on this feature can be found [here](#).

Listing 66. Access the properties inside your Twig templates.

```
{% set color = properties.styles['primary-color'] ?: '#ff00ff' %}
```

Listing 67. Use the `bsiProperty()` helper inside your templates.

```
@color: bsiProperty('styles.primary-color');
```

5.7. Build Configuration

In order to compile, it's required to add your template to the build configuration in the `webpack.config.js` file at the project root folder.

Listing 68. Content of the `webpack.config.js` build configuration file.

```
const path = require('path');

const {BuildConfig, ModuleConfig, WebpackConfigBuilder, Version, DesignType} =
require('@bsi-cx/design-build');

module.exports = WebpackConfigBuilder.fromConfigs(
  new BuildConfig()
    .withName('landingpage') ①
    .withVersion('1.0.1-alpha') ②
    .withTargetVersion(Version.CX_1_3) ③
    .withDesignType(DesignType.LANDINGPAGE) ④
    .withRootPath(path.resolve(__dirname, 'templates', 'landingpage')) ⑤
    .withPropertiesFilePath(path.resolve(__dirname, 'properties.js')) ⑥
    .withModules(
      new ModuleConfig() ⑦
        .withPath('main.js')
        .withName('main')));
```

- ① The name of this template.
- ② The current version of this template, optional but recommended.
- ③ The target BSI CX version. The build is capable of converting a design to the required format of various CX versions.
- ④ What type of design is your template?
- ⑤ The path to your template root folder. This is where your `design.js` is located.
- ⑥ The path to your `properties.js` file.
- ⑦ Configure your Java Script modules here.

6. Help

6.1. Good practices

The following things have been established as good practice in some projects. However, this approach is by no means mandatory and may vary depending on the project and individual preferences.

6.1.1. Deleting or deprecating content elements

Over time, a design evolves and due to various reasons, a certain content element may become obsolete. A good practice to phase-out a content element is to deprecate an element in a first step instead of directly deleting it. Deprecating helps to inform the end user about a content element that should no longer be added to new contents while still present in existing ones. In addition, it makes manual content migrations easier (see [Refactoring content elements](#)). In a future release, when you are sure that the content element is no longer actively in use, you can completely delete the content element.

6.1.2. Refactoring content elements

While refactoring a content element, keep in mind that your changes to a content element should never break existing content. We strongly advice to test your refactored content element to content that was created under the old design version to make sure that you do not introduce breaking changes. Critical points are renamings, changing the order for element parts and big changes to the DOM structure. In spite of that, we do encourage you to refactor where necessary, remind you to test carefully. If a refactoring is not possible due to migration issues, adding a new content element and deprecating the preexisting one (see: [Deleting or deprecating content elements](#)) is a feasible option.

6.1.3. Creating accessible designs

To ensure accessibility in CX contents, several things should be taken into account when developing a design. They help to ensure that screen readers can read out the content in the best possible way and that people with impaired vision will not be disadvantaged.

6.1.3.1. Layout

- All functionalities (form input fields and control elements) should be accessible and operable using the keyboard (tab key). It must also be possible to exit all elements using the keyboard so that the user can navigate through all content.
- Every website has a correct language declaration. If the main language of the page is English, all texts should be in English. If the intention is to insert text in another language, then this should be marked up accordingly with the lang attribute in the correct language.
- Enlargement is possible in the browser, both with "Zoom" and by setting a user-defined font size
- To ensure that screen readers interpret and parse all page content correctly, HTML elements of a website must be implemented in accordance with their W3C specifications. The [W3C's](#)

[Markup Validation Services](#) can be used to check whether the source code of a website complies with the W3C specifications.

6.1.3.2. Styling

- The contrast ratio of the font color to the background color is at least 4.5:1
- The contrast ratio of the font color of large font and the background color is at least 3:1
- If color differentiation alone is used, e.g. for links in a text, the links have a contrast ratio to the surrounding continuous text of at least 3:1
- Each element must have a visible focus state that has a contrast ratio of 3:1 to the background or non-focused state
- No spaces to create spacing, but CSS instead

6.1.3.3. Elements

- For information-bearing images, the alternative text must describe the visual information
- Headings are marked with the heading element (h1, h2, ... , h6)
- The text size (font size) in web templates is defined in CSS in % or em
- Enumerations are formatted as lists (ul, ol, dl)
- Special text is formatted correctly, e.g. quotes as `<blockquote>` and `<cite>`
- Video content should have synchronous subtitles
- Data tables are formatted with the necessary markup, e.g. column, row and table headings are labeled and headings and summaries are present

6.1.3.4. Forms

- Labels are present and meaningful. Labels and associated form input fields are logically linked (implementation with *for* attribute for label elements). The label should be present prior to the associated input field (exception: checkboxes, radio buttons).
- Form fields may have an additional information text to describe what input is expected. If, for example, a certain format is expected, this should be described (e.g. *"Date format: DD.MM.YYYY"*). This text should be placed in front of the input field.
- Mandatory fields should be marked (e.g. with *). The mandatory field star must not be integrated via CSS, it should be placed directly in the label. The required attribute should be added to the input element.
- A mandatory field explanation should be provided (e.g. *"Please fill in all fields marked with *"*). The information must appear before the first mandatory field.
- Error messages should appear directly below the relevant input field. They should have the attributes *aria-live="assertive"* (or *aria-live="polite"*) and *role="alert"* so that they are read aloud by the screen reader as soon as they appear.

6.2. Troubleshooting

In this chapter, we are collecting some common issues design developers may face in their development process.

6.2.1. Design Upload

While uploading a design, BSI CX will parse the content of the ZIP file to ensure that the containing design is valid. If the parser detects inconsistencies, an error message is displayed. The following trouble shooting sections will help to correct the error.

TIP

It is helpful to to view the details of the error message by clicking on **Details**, if available.

WARNING

In Versions prior to CX 1.2.48, the error message may be cut off in the message box. Use `Ctrl + C` and paste the error message to any text file to view the full details.

6.2.1.1. Validation error: File is missing

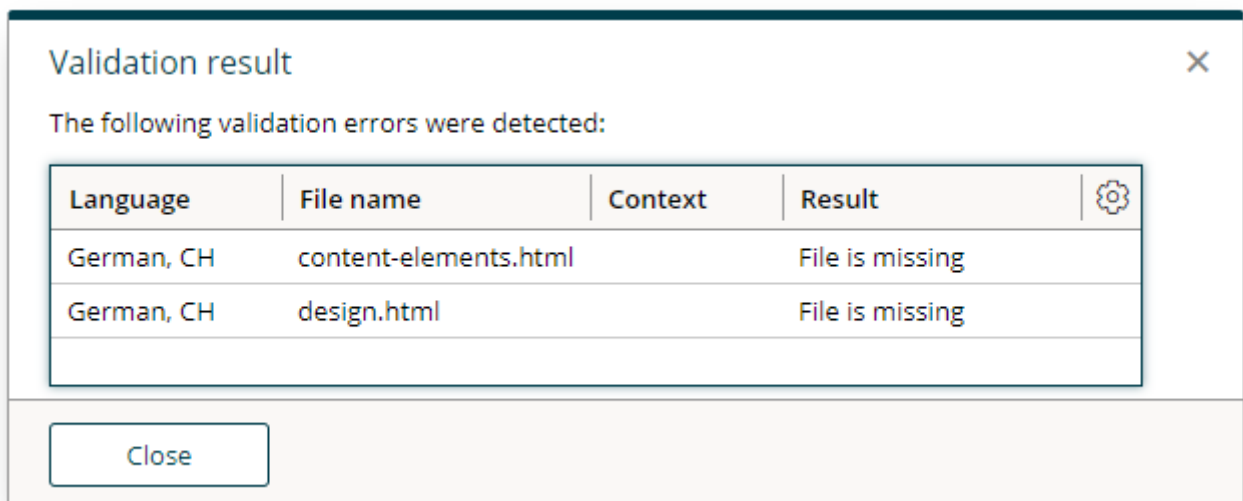


Figure 8. error message: file is missing

Your upload is missing a mandatory file. Please refer to [File format](#) where the mandatory files for your design type is described in detail.

6.2.1.2. Validation error: Invalid content element in 'data-bsi-dropzone-allowed-elements'

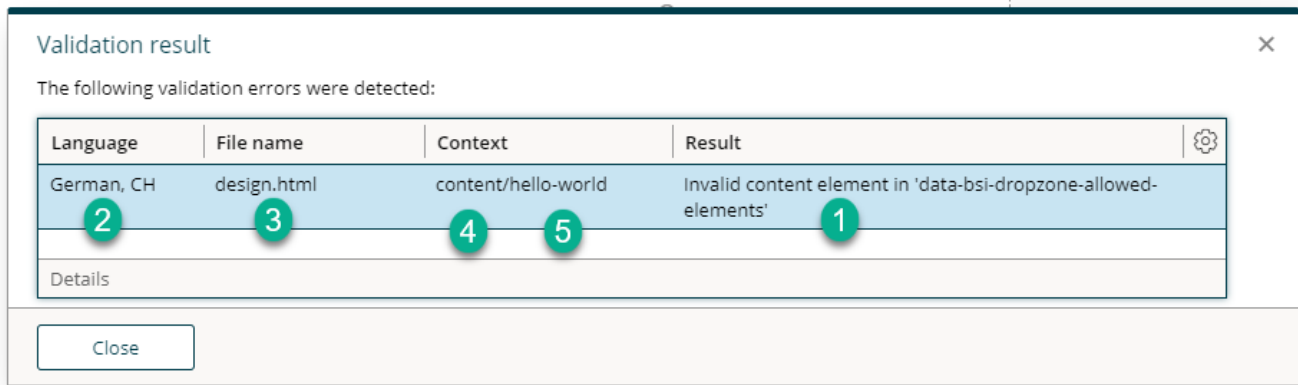


Figure 9. error message: invalid content element in 'data-bsi-dropzone-allowed-elements'

1. Error message
2. Affected ZIP file
3. Affected file (within the ZIP file)
4. Name of the *data-bsi-dropzone* in which the error causing content element is placed
5. The name of the content element. In this case, the content element name is missing

In one of your `data-bsi-dropzone-allowed-elements=''` definitions, there is an element listed which does not exist in the `content-elements.html` file. Common cases where this happens:

- A content element has been deleted but it is still referred by a dropzone
- A content element has been renamed and a dropzone is still referring to it under its previous name
- A new content element has just been implemented, but there is a mismatch in naming. Check again the name of your *data-bsi-element* and make sure it matches the name in the dropzone.

Listing 69. example of a badly referenced element in a dropzone

```
<!-- Design_de-CH.zip/design.html -->
<div data-bsi-dropzone="content"
      data-bsi-dropzone-allowed-elements="basic-cta hello-world">
  <!-- hello-world should be hello-text, see below -->
</div>

<!-- Design_de-CH.zip/content-elements.html -->
<div data-bsi-group="basic">
  <!-- in design.html, the element is incorrectly referenced under the name hello-
world -->
  <div data-bsi-element="hello-text" data-bsi-element-part="formatted-text">
    <p>Lorem ipsum</p>
  </div>
  <a data-bsi-element="basic-cta" href="#" data-bsi-element-part="link"
class="element basic-cta">Lorem ipsum</a>
</div>
```

6.2.1.3. Validation error: Unknown content element part

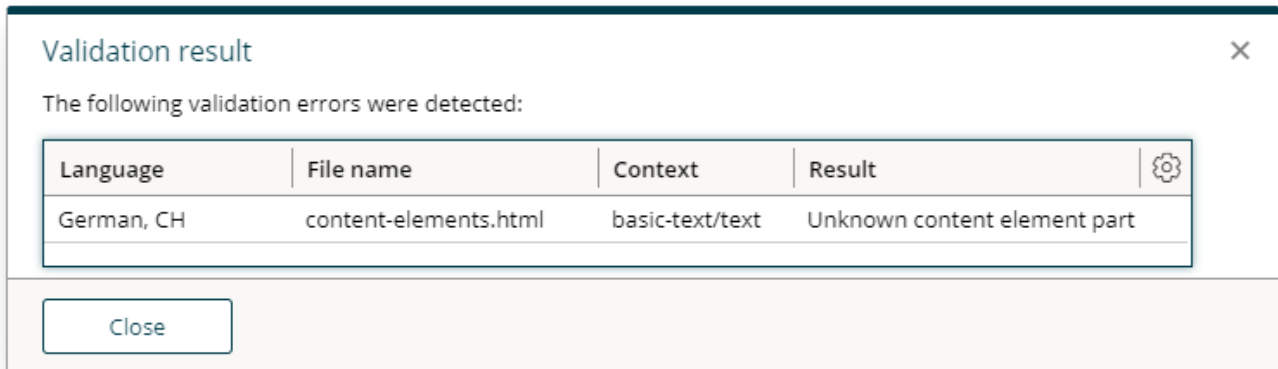


Figure 10. error message: unknown content element part

The specified `data-bsi-element-part` is not supported by CX. For a list of supported element parts, visit the [Parts](#) documentation chapter.

NOTE

In the example above, a *text* element part is specified. According to the documentation, it must either be *plain-text* or *formatted-text*. Thus, it is unknown to CX.

6.2.1.4. Validation error: Dropzone contains sample content element that doesn't match the structure of the reference content element

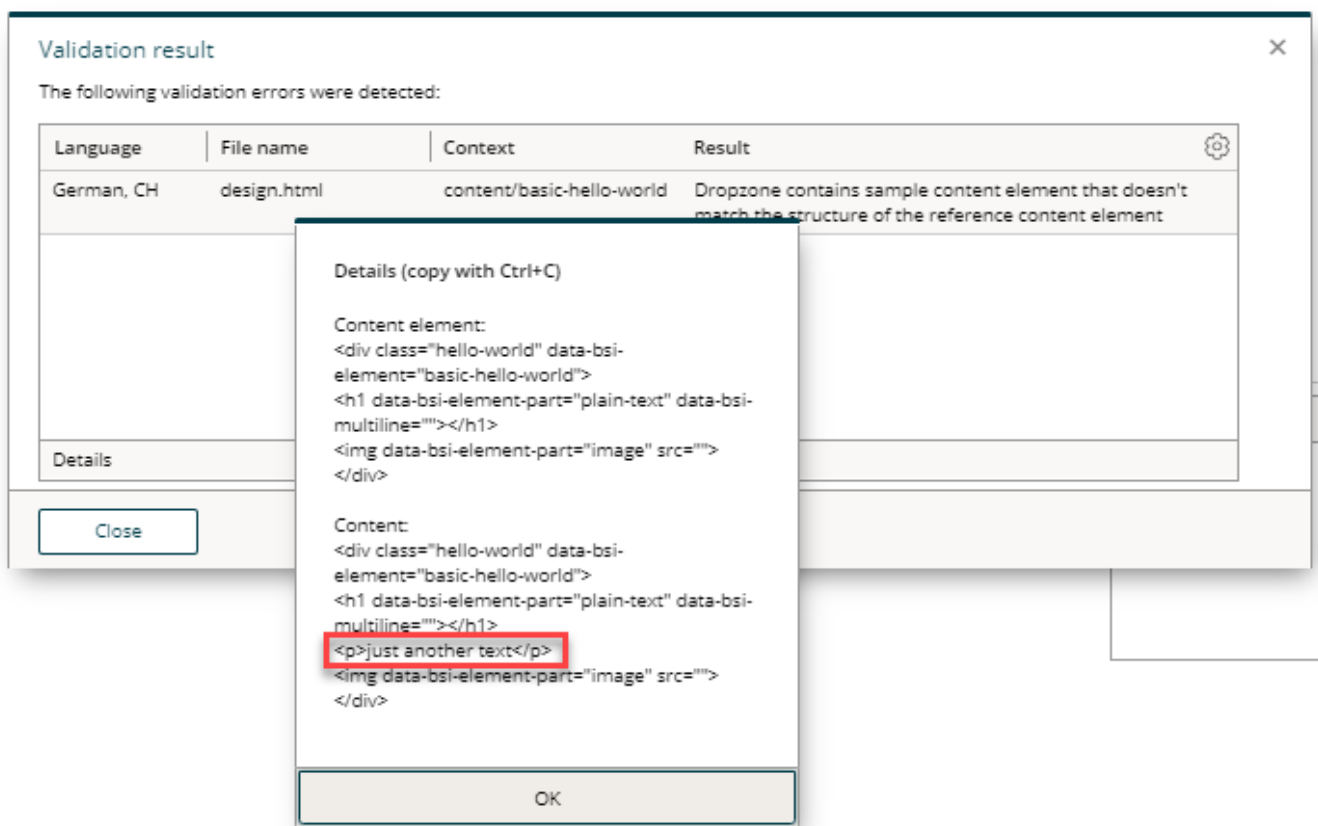


Figure 11. error message: Dropzone contains sample content element that doesn't match the structure of the reference content element

A discrepancy was discovered when using a content element as sample content. The example must have the same HTML structure as the specified content element.

TIP

A comparison of the two elements can be displayed by clicking on **Details**.

Listing 70. example of a sample content that does not match the specified content element

```
<!-- Content Element: Specification of the element (content-elements.html) -->
<div class="hello-world" data-bsi-element="basic-hello-world">
  <h1 data-bsi-element-part="plain-text">Placeholder</h1>
  
</div>
<!-- Content: sample content (design.html or content-elements.html) -->
<div class="hello-world" data-bsi-element="basic-hello-world">
  <h1 data-bsi-element-part="plain-text">Hello World, how are you?</h1> ①
  <p>just another text</p> ③
   ②
</div>
```

- ① OK: deviating text within an editable area (here: plain-text element part)
- ② OK: deviating attribute, which can be edited from CX for this type anyway
- ③ NOT OK: adding, manipulating or removing DOM nodes is not allowed

6.2.1.5. Validation error: Dropzone contains sample content that are not content elements

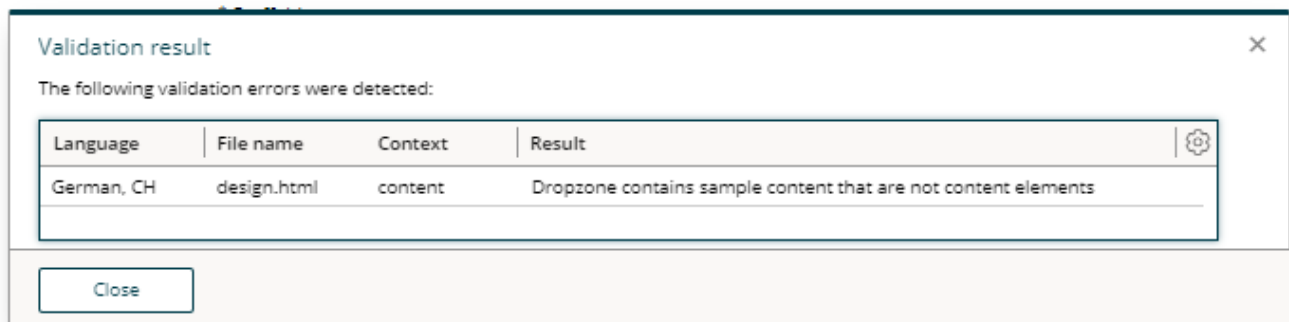


Figure 12. error message: Dropzone contains sample content that are not content elements

A dropzone can contain 0...n content elements. Only content elements are allowed as direct child nodes of dropzones. Therefore, HTML that is not part of a content element must not be placed in a dropzone.

Listing 71. example of a detached paragraph that must not be placed in a dropzone

```
<div data-bsi-dropzone="content" data-bsi-dropzone-allowed-elements="title-h1 title-h2
title-h3 basic-text basic-image basic-cta">
  <h1 data-bsi-element="title-h1" data-bsi-element-part="plain-text" class="element
title-h1">Lorem ipsum</h1>
  <p>The paragraph surrounding me is not part of a content element and therefore
must not be placed in the dropzone 'content'.</p>
</div>
```

6.2.1.6. Validation error: Part uses tag X but requires one of the following: Y,Z

Figure 13. error message: Part uses tag div but requires one of the following: a,img

Make sure to use proper HTML tags for your element parts. Consult the content element [Parts](#) documentation for the validation error causing element.

In the example above, this would be the [Image](#) content element [Parts](#) documentation.

Listing 72. example of an incorrect HTML tag for a certain element part and how to fix it.

```
<!-- Invalid: uses <div> which is not an image -->
<div data-bsi-element-part="image" src="img.png" alt=""></div>
<!-- Correct way: use an <img> tag -->

```

6.2.1.7. Validation error: The X tag must contain exactly one Y tag

Make sure to use proper HTML tags for your element parts. Consult the content element [Parts](#) documentation for the validation error causing element.

6.2.2. Design Update

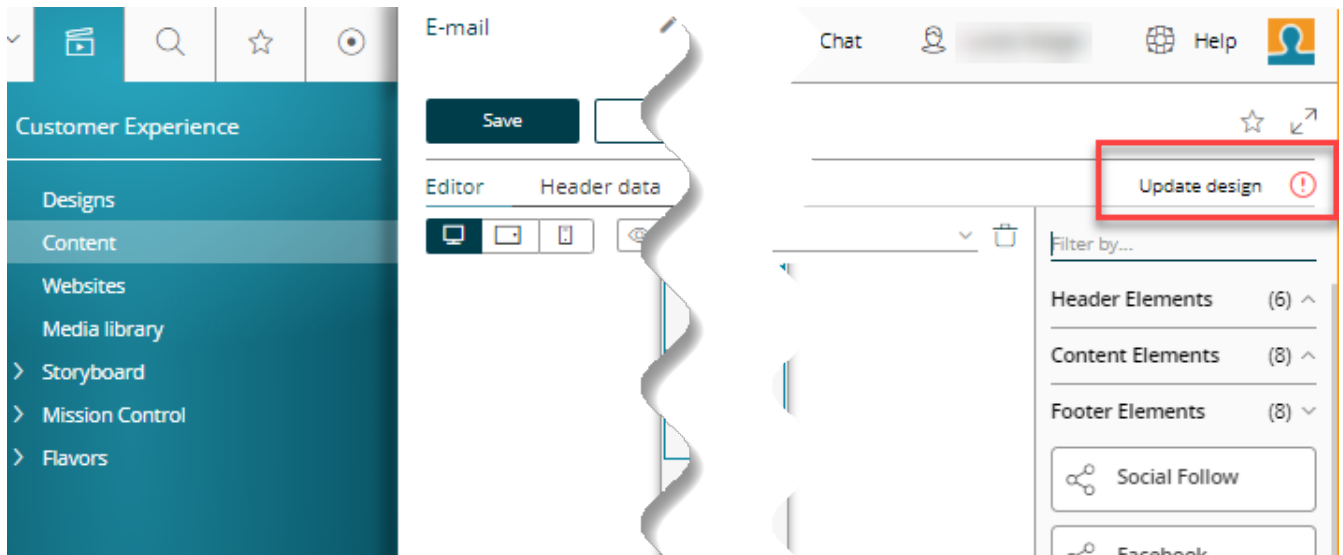
To update existing content onto the most recent version of a design, the content must be explicitly updated by clicking the **Update Design** button in the content editor. Success or failure of such a design update is indicated by a message box.

After an update, we advise to visually cross-check the content to make sure that a change in the design has not broken your content. In case if you are seeing content that does no longer render properly, your recent design version has introduced a flaw. In that case, you can revert the design update by clicking on the cancel button in the content editor.

Be reminded that once an updated content is saved, there is no way to go back to the older design version, unless you explicitly create a new version by selecting the respective checkbox next to the save button before saving the content.

6.2.2.1. Design Validation

After hitting the *design update* button or after a modification of your content, you may see a validation error indicated in the top-right corner.



the red exclamation mark indicates a validation error on design update

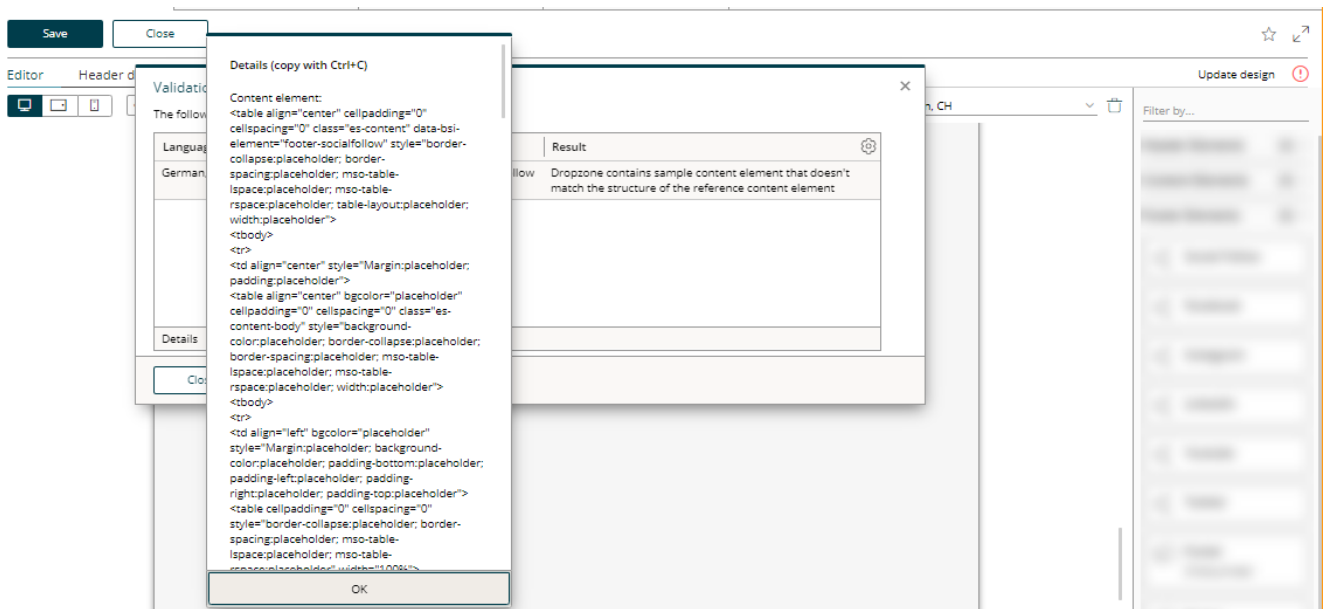
In such a case, there is an issue in the design used by your content. The most typical error messages related to design issues are described in the following chapter. You may also see some that are already described in the [Design Upload](#) section.

6.2.2.2. Content does not match the structure of the referenced content element

The structure of one of your content elements that is being used right now in your content does no longer match it's specification.

TIP

It is helpful to to view the details of the error message by clicking on **Details**, if available.



viewing validation error details

WARNING

In versions prior to CX 1.2.48, the error message may be cut off in the message box.

TIP

Use **Ctrl** + **C** and paste the error message to any text file to view the full details.

The details of such an error message always have the same structure:

Content Element:

```
<div>....</div>
```

Content:

```
<div>.....</div>
```

If the difference is difficult to find, we recommend using a diff tool to compare the concrete content with the content element specification.

The root cause of a difference between a concrete content and the content element specification may be one of the following:

JavaScript

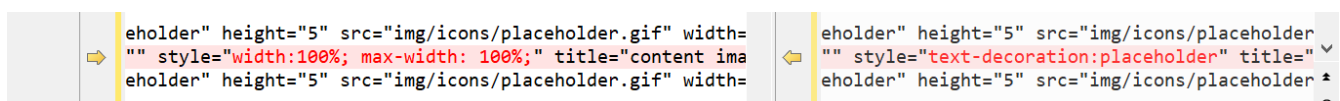
You have a JavaScript running that is manipulating your content element. To prevent your content from being modified through a script, make sure that you update your `<script>` tags in your *design.html* file so that the scripts will only be loaded if your content is **not** being edited right now in the content editor. Removing a `<script>` tag from the DOM if the content is being edited can be achieved by adding `data-bsi-remove-if='draft'`.

Listing 73. removing a script in draft (edit) mode

```
<script data-bsi-remove-if="draft" src="your-javascript that manipulates-the-DOM.js"
></script>
```

Duplicate attributes

If you compare the specification of the content element to the actual content and the difference is somehow related to a CSS value that is being replaced by the term **placeholder**, then you might have defined the error causing HTML attribute twice.



Listing 74. invalid HTML will lead to issues

```
<!-- Invalid HTML: the attribute 'style' is defined twice -->


<!-- By merging the 'style' attribute, we can fix the issue above -->

```

Usage of invalid HTML

Some browsers detect and try to *fix* invalid HTML. Therefore, the DOM structure of the HTML is modified by the browsers, whereas CX still expects the specified structure.

Listing 75. invalid HTML will lead to issues

```
<!-- Invalid HTML, as defined in content-elements.html -->
<table data-bsi-dropzone="table">
  <div data-bsi-element="text">...</div>
</table>

<!-- As soon as the HTML is dropped in CX, the browser will 'fix' the HTML, leading to
a different HTML for the concrete content -->
<table data-bsi-dropzone="table"></table>
<div data-bsi-element="text">...</div>
```

Inline SVG (older CX versions)

WARNING | This section applies to installations prior to CX 1.2.48.

Inline SVG is not supported in older CX versions. As a workaround, you may save the SVG to a file instead and include it with an `` tag. If it is a requirement to render the SVG inline, some projects have successfully used the Open-Source library [svg-inject](#) as a fallback.

6.2.3. Dropzone Issues

If you face issues related to dropzones (a dropzone does not allow to drop your element, the element is dropped in a different place, ...), you may find a solution in this section.

6.2.3.1. Element is not dropable or being dropped in another place

WARNING | This section applies to installations prior to CX 1.2.46.

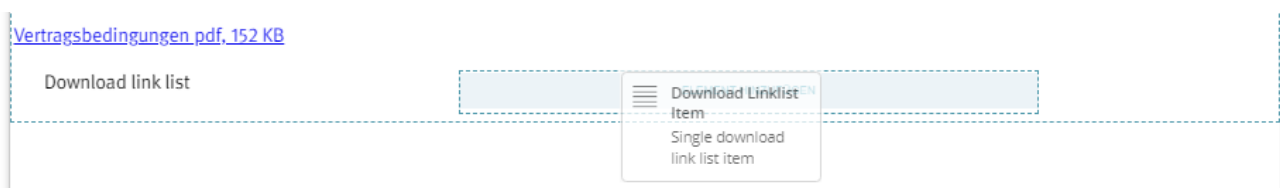


Figure 14. element is being dropped outside of the dropzone (or nowhere)

This may happen if a dropzone name is duplicated. Dropzones must have unique identifiers in older versions of BSI CX. Make sure that a unique name is specified for each usage of `data-bsi-dropzone`.

Listing 76. *data-bsi-dropzone uniqueness violated*

```
<!-- design.html -->
<div class="main-dropzone" data-bsi-dropzone="content" ①
  data-bsi-dropzone-allowed-elements=".....">
  <!-- ... -->
</div>

<!-- content-elements.html -->
<div class="image-area" data-bsi-dropzone="content" ②
  data-bsi-dropzone-allowed-elements="download-item"></div>
```

- ① A dropzone named 'content' is being defined properly
- ② Violation: another dropzone with the same name is defined again. Solution: change `data-bsi-dropzone` to another value, e.g. `data-bsi-dropzone="image-area-content"`

6.2.4. Design Creator Issues

6.2.4.1. Why is the Design Creator End of Life?

We've built a next generation Design Build that is Open-Source and based on widely used tech stack. You may find more information on [GitHub](#).

6.2.4.2. Is there any detailed documentation for the CX Design Creator

Unfortunately there is no detailed documentation. We do no longer invest into the *CX Design Creator* as our sucessor, the [CX Design Build](#), is ready to use.

6.2.4.3. HTML Comments / MSO Tags missing after running the CX Design Creator

If you are using the *CX Design Creator* and your HTML in the `design.zip` is missing all HTML comments and therefore all outlook specific code is missing, change the `tidy` property to `false` in the `compiler.yml`:

```
defaults:
  twig:
    comments: true
    tidy: false
```

6.2.5. Content Editor Limitations

6.2.5.1. Using CSS viewport units

If you want to use CSS viewport units, e.g. `viewport-height` `vh`, please note the following:

The content displayed in the content editor is embedded in an `iframe`. When elements defined by a CSS viewport unit are placed inside an `iframe`, the size of the `iframe` is used as a reference, not the size of the screen as desired.

Therefore, the CSS classes `bsi-ce-edit-mode` and `bsi-ce-preview-mode` (see [Content editor specific CSS](#)) should be used to define a fixed height/width as alternative to the viewport height/width for proper display in the content editor and preview mode. It is also important to note that this fixed height/width should be set as minimum size of the element with the CSS viewport unit. Otherwise CX will calculate the size of the content editor incorrectly.

See the following example:

Listing 77. CSS of an element using viewport unit

```
.header-image {  
    height: calc(100vh - 32px);  
    min-height: 800px !important;  
}  
.bsi-ce-edit-mode .header-image,  
.bsi-ce-preview-mode .header-image {  
    height: 800px !important;  
}
```