What is yeet?

INFO

The term "yeet" is a *US* slang expression. It is used to express excitement, joy, happiness, exuberance and approval.

The Citizen Developer is a user of a low-code platform to create business applications on his own without great programming skills. With yeet the user gets an uncomplicated access to own digital solutions.

Signs of a new era: significant cost reduction for new development Increased productivity of your team through steep learning curve and accelerated software development. Low-code solutions fuel new ideas on the one hand, and gaps can finally be closed on the other. Previously, the budget may not have been sufficient. Or no suitable software was found. Now business ideas can simply be tried out. Smart and mobile - for developers and users alike Regardless of the operating system: An Internet browser is all that is needed to use it.

Whether on a slim phone with Google Chrome, Firefox on a heavy desktop computer or a tablet with Safari: yeet feels at home everywhere!

> *"With agile user feedbacks we validate valid results"*
>
> — Miqueas Cramer, front-end developer at vectorsoft

Limitless possibilities: Low-Code and Classic Code in combination With yeet, existing conzept 16 solutions can be seamlessly implemented and further developed.

As a conzept 16 developer, you can combine the creation of a yeet user interface (UI) with your existing business logic from conzept 16 with the first expansion stage (2022). yeet Logo | vectorsoft What is Low-Code? In Low-Code you mainly work with visual tools. Low-Code platform yeet | copyright vectorsoft

When programming becomes necessary, yeet has a special solution ready. Whether Python, Typescript or JavaScript: You just choose your favorite language! What is yeet? yeet is a software development program of the latest generation. You call the frontend in the browser. The most important area is the app designer for visual programming. yeet | the origin The term "yeet" is a US-American slang expression. It is used to express excitement, joy, happiness, exuberance and approval. Create your own app without programming experience With yeet and Low-Code your own IT project becomes reality: The result is an executable desktop application, web app and mobile app in one.

With yeet's Flexbox, you develop in Responsive Design right from the start. Your application automatically adapts to all end devices. How yeet builds on conzept 16... yeet is based on a three-layer architecture: frontend with the yeet App-Designer, backend with the Application Services and the database layer with conzept 16.

The integration of the conzept 16 database is an important component for the successful adoption or extension of existing conzept 16 solutions. With an open interface standard At the backend level, a standardized REST interface exists in the form of OData. The HTTP-based protocol developed by Microsoft regulates secure and fast data access between different sources and software systems. vectorsoft: Interfaces from yeet

Especially the web-based access to existing conzept16 databases is possible by means of this modern architecture.

Edit this page

*Last updated by **Carl Ahrens***

Next
Getting Started »

Copyright © 2024 YeetDocs, vectorsoft AG.

Getting Started

Getting Started with yeet

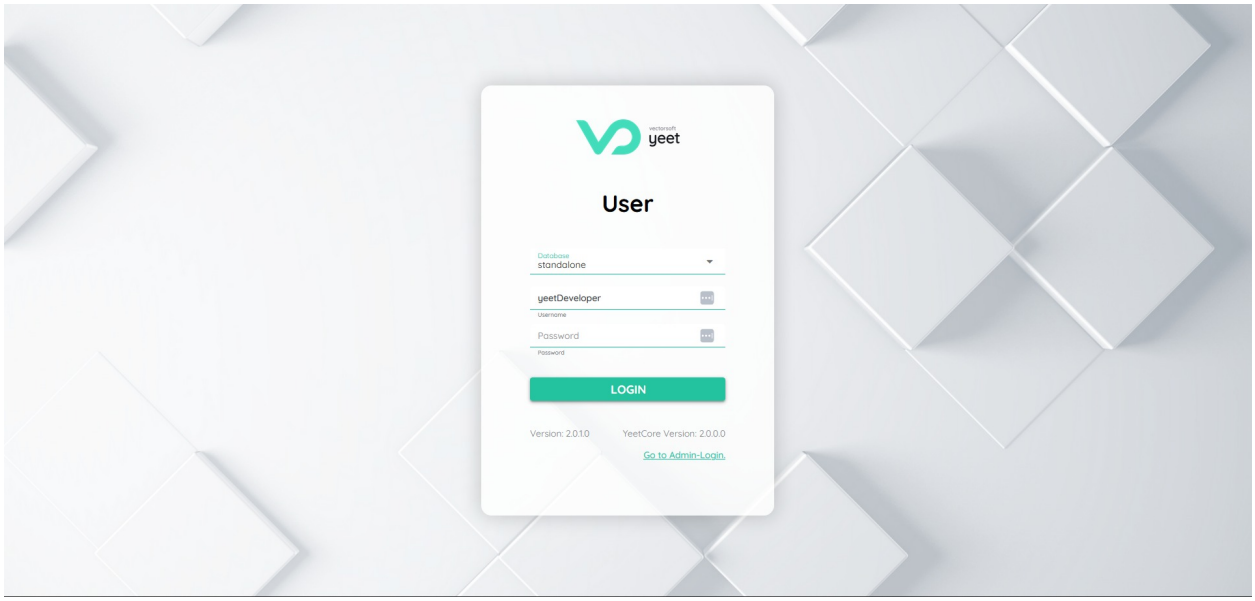
To get started with yeet, a yeetCore must first be installed on a server/local machine. If this has not been done yet, please refer to the following page.

Login

Zum überprüfen ob deine yeetCore installation erfolgreich war, kannst du deine Domain ins Input eintragen und dann den erstellten Link besuchen.

https://

Ist die Loginseite erfolgreich im Browser geöffnet können wir mit unserem ersten yeet-Projekt starten. Dazu wählen wir die von dir eingestelle Datenbank aus und loggen uns mit folgenden Daten in yeet ein.



User: yeetDeveloper
Passwort: <kein Passwort>

Wir starten im **Cockpit** und können nun entscheiden mit welchem Bereich unserer WebApp wir starten wollen.

Projects
Hier findest du alle Projekte die in der Datenbank erstellt worden sind. Ein Projekt beinhaltet alle Daten für die Web-Oberfläche. Dazu gehören Pages, UI-Snippets, Clipboards und Popups.

Themes
Jedes Projekt können Themes zugewiesen werden. Sie bestimmen den Basisstyle aller Komponenten in einem Projekt.

Frontend
Hier kommst du in den UI-Designer mit dem zuletzt geladenen Projekt.

Data
Über Data kannst du das Datenmodel aus Entities erstellen und mit Dateneinträgen füllen. Die Daten kannst du dann über den UI-Designer mit Komponenten verknüpfen.

Business Logic
Sollen komplexe Funktionen auf den Daten ausgeführt werden, können hier Server-Funktionen mittels Javascript hinterlegt werden.

Access Control
Weitere Test-User oder Rollen können hier, für deine Projekte angelegt werden.

Help-Center
Brauchst du Support oder andere Hilfe wirst du hier fündig

Settings
Alle Einstellungen bezüglich yeet findest du untern diesem Punkt.

Edit this page
*Last updated by **Felix***

Previous
« What is yeet?

Next
Installation »

More

Blog

GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

yPage

ExampleBasicConfigExample1

Description

The yPage is an element that can contain other elements as the name implies. It is used to structure the page and store elements.

Some uses for the yPage are:

- Theming

Properties

Through its various properties the **yPage** can be configured to suit your needs. The display below provides you with an overview of all the **yPage** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.

- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Events

▶ META

▶ DISPLAY

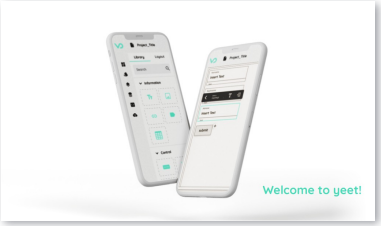
▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yPage in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Background



Cover

style	> background	> bg-image	https://www.vectorsoft.de/content/uploads/2021/09/-1024x768.png
style	> background	> bg-size	cover

Repeat

style	> background	> bg-image	https://www.vectorsoft.de/content/uploads/2021/09/-1024x768.png
style	> background	> bg-size	contain
style	> background	> bg-repeat	repeat

Alignment

This section shows the visualization of example related to the flexbox system.

PageAlignmentExample1

PageAlignmentExample2

Row Column Wrap

style	> alignment	> flex-direction	row
style	> alignment	> justify-content	center

INFO: SCROLL BEHAVIOR

> If the content exceeds the parent element itself, a scrollbar appear. When needed, this scrollbar can be removed while keeping scrollability, when toggling **show-scrollbar**.

Theming

How to:

You can use the yPage to theme the content inside.

1. Navigate to **"Events"** inside the Detail Panel and modify **"EvtInit"**.
2. Proceed to set the SubTheme based on a value inside a Clipboard.

Code example:

In this example the user can adjust the current theme by reading a property inside a Clipboard. Preparatory work is creating a Clipboard and setting the property inside.

EvtInit

```
// function onPage_1_EvtInit (apiObject, component, eventData) {

const useDarkTheme = await apiObject.Clipboards.propertyRead(0, "useDarkTheme");

if (useDarkTheme) {
  component.set("generic", "meta", "SubTheme", "darkTheme");
} else {
  component.set("generic", "meta", "SubTheme", "lightTheme");
}

// }
```

Edit this page

*Last updated by **Felix***

Previous
« UI Components

Next
yPopup »

Twitter

More

Blog

GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

yPopup

ExampleBasicConfigExample1

Description

The yPopup is an element that can contain other elements as the name implies. It is used to structure the page and store elements.

Some uses for the yPopup are:

- Creating SourceLink open/close
- Using submit and close
- Using modal popup for multiple pages

Basic Structure

In the following the basic structure of the yPopup shall be explained. For the general structure of a yComponent please visit the YBase-components basic-structure.

@Note insert image here

The yPopup as seen above can contain a header and footer and always includes content inbetween.

Header

The header is modified to be more than display type normal. It contains a title and an icon on the left

side and a close button on the right.

Content

The content is the actual center of the component itself. It basically represents a container which can be designed and filled like a yPage.

Footer

The footer is a specific submodule of the yPopup with display type modal. It contains two action buttons on the right that can be modified if needed. These two buttons represent a cancel and a submit button which can trigger actions.

Properties

Through its various properties the **yPopup** can be configured to suit your needs. The display below provides you with an overview of all the **yPopup** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Links Events

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

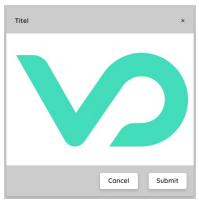
In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yPopup in ways that deviate from its standard behavior, as defined by yBase. This section

aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Types

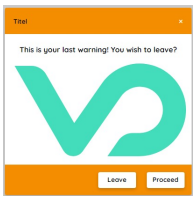
Modal

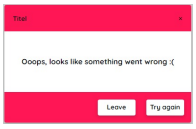
The dialog of type modal is a popup with further actions that can be executed with the buttons in the footer. Furthermore it is persistent and always appears in the center of the application with a backdrop.



PopupModalExample2

PopupModalExample3





Normal Info Success Warning Error

style	> popup	> popup__display-type	modal
style	> popup	> popup__display-state	normal

Persistence, Timed, Click

- The **persistence** popup includes a header and the idea that it can't be closed by clicking away.
- The **timed** popup includes a header and closes automatically after the set ms value for **duration**.
- The **click** popup includes a header and can be closed by clicking away.

PopupPersistenceExample1

PopupPersistenceExample2

PopupPersistenceExample3

PopupPersistenceExample4

Normal Info Success Warning Error

style	> popup	> popup__display-type	persistence/timed/click
style	> popup	> popup__display-state	normal

Background

Cover

style	> background	> bg-image	https://www.vectorsoft.de/content/uploads/2021/09/-1024x768.png
style	> background	> bg-size	cover

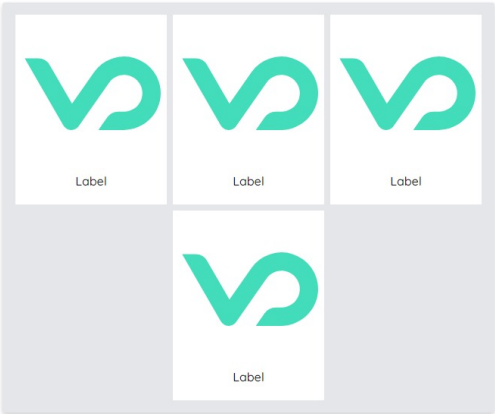
Repeat

style	> background	> bg-image	https://www.vectorsoft.de/content/uploads/2021/09/-1024x768.png
style	> background	> bg-size	contain
style	> background	> bg-repeat	repeat

Alignment

PopupAlignmentExample1

PopupAlignmentExample2



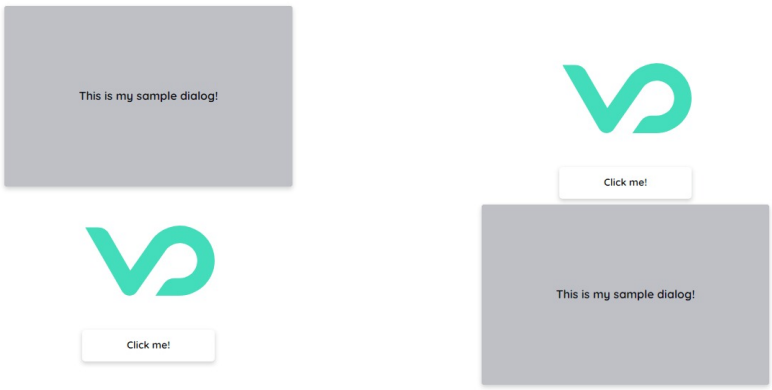
Row Column Wrap

style	> alignment	> flex-direction	row
style	> alignment	> justify-content	center

> **INFO: SCROLL BEHAVIOR**
>
> If the content exceeds the parent element itself, a scrollbar appear. When needed, this scrollbar can be removed while keeping scrollability, when toggling **show-scrollbar**.

Positioning

To position a yPopup, you have to choose a position inside the PositionPicker.



PopupPositioningExample3

Inner top center	Outer bottom center	Inner center center	
style	> popup	> type	normal
style	> popup	> displayPosition	inner-top-center
style	> popup	> sourceElementParent	YOUR_PAGE
style	> popup	> sourceElementChild	YOUR_ELEMENT

> **INFO**
>
> To make the popup persistent, which means it cannot be close by clicking away, toggle **persistent** to `true`.

Creating SourceLink open/close

When created and styled, you obviously want to display your yPopup.

How to:

1. Within the Detail Panel, proceed to the **"Style"** section and open **"Popup"**.
2. Change the **"display-type"** to modal or any other type.
3. Create a yPage with a yButton component in your application.
4. Open your yPopup again.
5. Within the Detail Panel, proceed to the **"Links"** section and unfold **"Open"**.
6. Create a new SourceLink to the yButton on the other yPage.

> **INFO**
> The same procedures can be done with any other trigger element and also applys for the **"Close"** SourceLink.

Using submit and close event

How to:

You can use the yPopup as an modal dialog and react on user interactions such as **submit** and **close**.

1. Within the Detail Panel, proceed to the **"Style"** section and open **"Popup"**.
2. Change the **"display-type"** to modal.
3. The proceed to the **"Event"** section and unfold **"Input"** and open **"EvtSubmit"**.

Code example:

In this example we have a simple yPopup containing a minified calculator that can only add two values. When the user submits, we read the two input values which should be added and alert the result.

EvtSubmit

```
// function onPopup_1_EvtSubmit (apiObject, component, eventData) {

const inputA = apiObject.ui.getObject(1, "y-input");
const inputB = apiObject.ui.getObject(2, "y-input");

const num1 = inputA.get("style", "content", "inputNumber");
const num2 = inputB.get("style", "content", "inputNumber");

alert("result: ", num1 + num2);

// }
```

Using modal popup for multiple pages

How to:

You can use the yPopup universally on multiple pages inside your project.

1. Open the Detail Panel and proceed to **"Style"** and unfold the group **"Popup"**.
2. Change the **"display-type"** to modal.
3. Create a yPage with a yButton in your application.
4. Within the Detail Panel of the yButton, proceed to the **"Event"** section and open **"EvtClicked"**.

Code example:

In this example the user can call the popupOpen function to open a certain yPopup, when a yButton is

clicked.

EvtClicked yButton

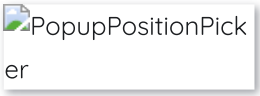
```
// function onButton_5_EvtClicked (apiObject, component, eventData) {

apiObject.popup.popupOpen(1, "y-popup");

// }
```

Special Picker

Position Picker

The position picker is a special picker in the Detail Panel, which enables the positioning of popups.

> **INFO**
> This doesn't affect popups of type modal, because they are always centered!



The picker lets you choose between 25 different positions for elements and 9 positions for pages. The position you choose, is always related to the element by the left bottom corner of the popup itself. Therefore you need to choose a **sourceElement** first.

Edit this page

*Last updated by **Felix***

Previous
« yPage

Next
yContainer »

More

Blog

GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

yContainer

ExampleBasicConfigExample1

Description

The yContainer is an element that can contain other elements as the name implies. It is used to structure the page and store elements.

Some uses for the yContainer are:

- Theming

Properties

Through its various properties the **yContainer** can be configured to suit your needs. The display below provides you with an overview of all the **yContainer** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.

- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Container Events

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

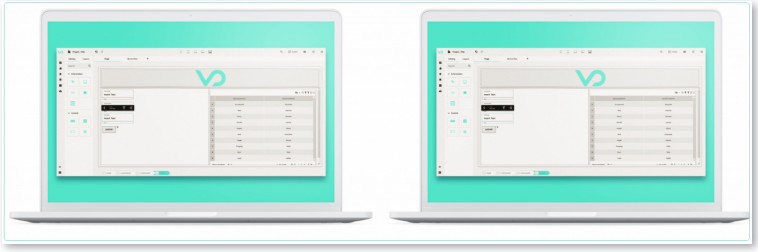
In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yContainer in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Background

Contain

style	> background	> bg-image	https://www.vectorsoft.de/content/uploads/2021/09/laptop-1024x633.png

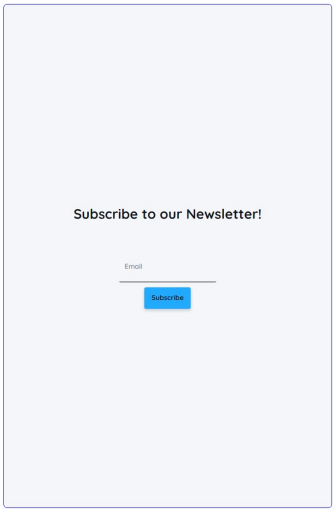
style	> background	> bg-size	contain
style	> background	> bg-repeat	no-repeat

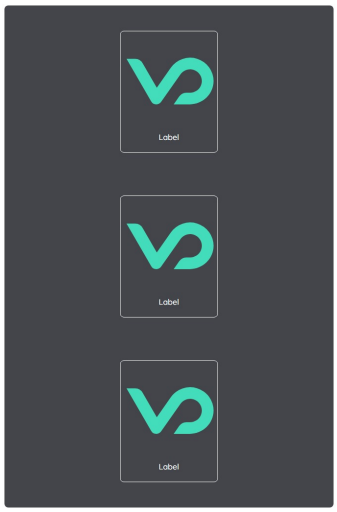


Repeat x

style	> background	> bg-image	https://www.vectorsoft.de/content/uploads/2021/09/laptop-1024x633.png
style	> background	> bg-size	contain
style	> background	> bg-repeat	repeat-x

Alignment





Column Space around Wrap

style	> alignment	> flex-direction	column
style	> alignment	> justify-content	center
style	> alignment	> align-items	center

INFO: SCROLL BEHAVIOR

If the content exceeds the parent element itself, a scrollbar appear. When needed, this scrollbar can be removed while keeping scrollability, when toggling **show-scrollbar**.

Theming

How to:

You can use the yContainer to theme submodules inside your page.

1. Navigate to **"Events"** inside the Detail Panel and modify **"EvtInit"**.
2. Set SubTheme based on a selection from yToggle.

Code example:

In this example the user can adjust the current theme by toggling, which changes the SubTheme of the container every time the container is reloaded. We read the value from a toggle which needs to be created.

EvtInit

```
// function onContainer_6_EvtInit (apiObject, component, eventData) {

const useThemeToggle = apiObject.ui.getObject(1, "y-toggle");
const useDarkTheme = useThemeToggle.get("style", "thumb", "value");

if (useDarkTheme) {
  component.set("generic", "meta", "SubTheme", "darkTheme");
} else {
  component.set("generic", "meta", "SubTheme", "lightTheme");
}

// }
```

Edit this page

*Last updated by **Felix***

Previous
« yPopup

Next
yImage »

More

Blog

GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

yImage

y-Image basic config normal

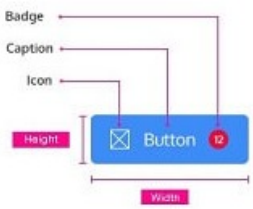
Description

The yImage is aclickable element with different optical styles, used to indicate its different states. The yImage can be used to display image content and to trigger a user action. For example:

- Using external database
- Setting image via SourceLink
- Redirect on click

Basic Structure

In the following the basic structure of the yImage shall be explained. For the general structure of a yComponent please visit the YBase-components basic-structure.



The yImage as seen above contains a label.

Label

The label contains the text inside the image. It is normally in the center of the component, but can be shifted horizontally or vertically. By changing the inner padding of the image, the label gets moved too.

Image

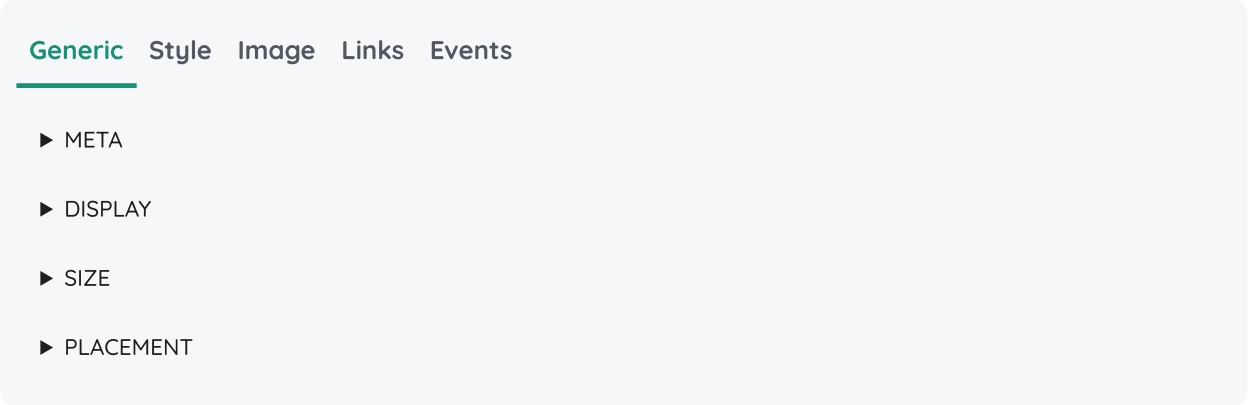
The image can be used to show the image itself. Multiple style adjustments can be executed to customize the appearance.

Properties

Through its various properties the **yImage** can be configured to suit your needs. The display below provides you with an overview of all the **yImage** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.



Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yImage in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Image Elevated Image Flat Image Outlined Image Rounded Image Plain

Elevated | Flat | Outlined | Rounded | Plain

generic	> size	> minWidth	150px
generic	> size	> minHeight	150px
style	> background	> bgColor__normal	#44ddbb
style	> shadow	> boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px

Label

The label can be used to show text inside the yImage. You can only have one label in a yImage. If you want to customize it, you can do this with the TypographyPicker in Toolbar or under the style menu in the Detail Panel.

Image Bottom Center Image Bottom Right

Bottom Center | Bottom Right

generic	> size	> minWidth	150px
generic	> size	> minHeight	150px
style	> label	> label__fontFamily__normal	Roboto
style	> label	> label__fontSize	22px
style	> label	> label__fontWeight__normal	bold
style	> label	> label__fontColor__normal	#FFFFFF
style	> label	> label__labelPositionY	flex-end
style	> label	> label__fontAlign	center

Image

Rotation | Positioning

generic	> size	> minWidth	150px
generic	> size	> minHeight	150px
style	> image	> image__url	https://images-na.ssl-imag amazon.com/images/I/61s
style	> image	> image__bgSize	cover
style	> image	> image__transform	45deg
style	> border	> borderStyleTop__normal	solid
style	> border	> borderStyleRight__normal	solid
style	> border	> borderStyleBottom__normal	solid
style	> border	> borderStyleLeft__normal	solid
style	> border	> borderWidthTop__normal	2px
style	> border	> borderWidthRight__normal	2px
style	> border	> borderWidthBottom__normal	2px
style	> border	> borderWidthLeft__normal	2px
style	> border	> borderColorTop__normal	#E4E6EA
style	> border	> borderColorRight__normal	#E4E6EA
style	> border	> borderColorBottom__normal	#E4E6EA
style	> border	> borderColorLeft__normal	#E4E6EA

External database

When working with external ressources, setting an image based on a external source may be relevant.

How to:

1. Open the Detail Panel and proceed to the **"Style"** section to open **"Image"**.
2. Insert the external ressource database inside the **"src-url"** property.
3. Add the reference to your image inside **"url"**.

> **WARNING: LACKING SUPPORT**
>
> This feature is **not supported yet**!

Setting image via SourceLink

How to:

1. Create a yInput component in your application.
2. Select the created yImage and open the Detail Panel to proceed to the **"Links"** section and open **"Image"**.
3. Create a new SourceLink -> select the yInput you just created and choose **"input-string"** as an event.

Now each time you insert a url into the input, the image will render the picture behind this url.

Redirect on click

How to:

1. Navigate to **"Events"** inside the Detail Panel and modify **"EvtMouse"** inside the group **"Mouse"**.
2. Associate a custom JavaScript function to the yImage. This function will be invoked every time the image is clicked.

Code example:

You can add the following JavaScript code snippet to your application to implement a function for navigating with the yImage.

With this script, a click on the yImage will redirect the user to another page of your application.

> **INFO**
>
> For this to work, you need to add routes for your pages inside the navigations.

EvtMouse

```
// function onImage_2_EvtMouse (apiObject, component, eventData) {

// for redirecting inside my application
apiObject.system.router.push("YOUR PAGE ROUTE HERE");

// for redirecting to an external page
window.location.replace("YOUR EXTERNAL ROUTE HERE");

// }
```

Edit this page

*Last updated by **Wegener***

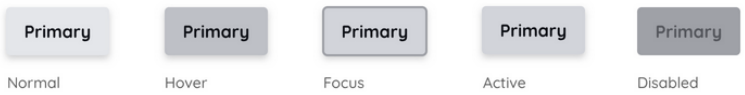
Previous
« yContainer

Next
yLinkButton »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yLinkButton



Description
Basic Structure
Properties
Usage
Variations
Label
Icon
Redirecting
Calling a custom javascript function
Interacting with Clipboard

Description

The yLinkButton is a clickable element with different optical styles, used to indicate its different states. The yLinkButton can be used to trigger a user action and to redirect to other pages. For example:

- Redirecting
- Calling a custom JavaScript function
- Interacting with Clipboards

Basic Structure

In the following the basic structure of the yLinkButton shall be explained. For the general structure of a yComponent please visit the **YBase-components basic-structure**.

The yLinkButton as seen above contains a label and a icon.

Label

The label contains the text inside the button. It is normally in the center of the component, but can also be displayed on the right or left site. By changing the inner padding of the button, the label gets moved too. You can find further explanation at the **Label Property**.

Icon

The icon can be used to show an icon inside the yLinkButton. You can have a maximum of two icons in a yLinkButton. The icon can be before or after the text and if you want to have only one yLinkButton with one icon, then you have to delete the text in the Label. You can find further explanation at the **Icon Property**. An icon is structured as you can see below:

Properties

Generic **Style** **LinkButton** **Events**

- META
- DISPLAY
- SIZE
- PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yLinkButton in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Button Elevated Button Flat Button Outlined Button Rounded Button Plain

Elevated Flat Outlined Rounded Plain

generic	> size	> minWidth	120px
style	> background	> bgColor__normal	#44ddbb
style	> shadow	> boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px
style	> label	> label__fontColor__normal	#FFFFFF

Label

LinkButton Label

Uppercase Label

generic	> size	> minWidth	150px
style	> label	> label__fontColor	#FFFFFF
style	> label	> label__fontFamily	'Roboto'
style	> label	> label__fontStyle	'italic'
style	> label	> label__fontSize	20px
style	> label	> label__fontWeight	'bold'
style	> label	> label__textTransform	'uppercase'
style	> label	> label	'Proceed'

Icon

LinkButton Icon Right LinkButton Icon Both LinkButton Icon Only

iconRight iconBoth iconOnly

style	> background	> bgColor__normal	#FF1654
style	> label	> label__fontColor__normal	#FFFFFF
style	> label	> label__fontWeight__normal	bold
style	> label	> label__fontAlign	center
style	> label	> label__textTransform__normal	uppercase
style	> icon	> icon__colorPrimary__normal	#FFFFFF
style	> icon	> iconPosition	iconRight
style	> icon	> iconRight	fas-fa-arrow-right

Redirecting

The button component enables you redirect to pages or extern URL each time it is clicked.

For instance, you could redirect to an official website to give further information about a topic your application handles.

How to:

1. Navigate to **"Component"** inside the Detail Panel and unfold the group **"Misc"**.
2. Choose a LinkTarget of your liking by inserting the relative route of your page or an external url.

Calling a custom javascript function

The yLinkButton component enables you to execute a custom JavaScript function each time it is clicked, similar to the yButton.

For instance, you could set up a function that alerts "Attention, redirecting!", before redirecting.

How to:

1. Follow the steps in Redirecting Example.

2. Open the editor under **"Event"**, unfold **"Input"** and open **"EvtClicked"**.
3. Associate a custom JavaScript function to the yLinkButton. This function will be invoked every time the button is clicked.
4. Include the provided JavaScript snippet below to implement an alert function on the button.

Code example:

You can add the following JavaScript code snippet to your application to implement a an alert function on the yLinkButton component.

With this function, each click alerts a message before redirecting.

EvtClicked

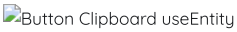
```
// function onLink-button_7_EvtClicked (apiObject, component, eventData) {

alert("Attention, redirecting!");

// }
```

Interacting with Clipboard

You can use the yLinkButton to control Clipboards. The yeetClipboard functionality provides an effective method for managing for example table entries. As example, you can set up a mechanism to delete a table entry by a simple button click.

How to:

1. Open the Detail Panel and proceed to the events tab and locate the **"Control"** section. Here, you'll need to enable the **"Commit-Create"** option. This step empowers the yLinkButton component with the capability to perform this functionality.
2. Create a new Clipboard and enable the **"Use Entity"** toggle.



3. Then you can click on **"Entity-Properties"** and edit the **"Entity"**. This Clipboard should be linked to the entity containing the table entry you wish to delete. This way you have to fill in the data, like in the picture below.

4. The _id under "Keys" should be the entry that you wish to delete.
5. Finally, go to the "Events", edit it and select the yButton as SourceLink-1 via the SourceLink Picker. Dont forget that the event has to be "commit-delete".

Events

Select Row

Sourcelink-1:

Commit Create

Sourcelink-1:

Commit Read

Sourcelink-1:

Commit Update

Sourcelink-1:

Commit Delete

Copyright © 2024 YeetDocs, vectorsoft AG.

Previous
« yImage

Next
yLabel »

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yLabel

Description
Basic Structure
Properties
Usage
Variations
Linking input elements
Displaying table data

Description

The yLabel component serves as a powerful tool primarily designed for displaying short text information within your application. It's an adaptable element that allows both static and dynamic content.

With the yLabel you can display user instructions, show real-time data updates, provide context for other components, or present calculated values within a form, among other applications.

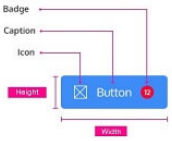
Some uses for the yLabel are:

- Linking input elements
- Displaying table data

> **TIP**
> Use yTextarea or yTextbox for handling longer texts.

Basic Structure

In the following the basic structure of the yLabel shall be explained. For the general structure of a yComponent please visit the **YBase-components basic-structure**.



Properties

Through its various properties the **yLabel** can be configured to suit your needs. The display below provides you with an overview of all the **yLabel** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Label Links Events

- META
- DISPLAY
- SIZE
- PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yLabel in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Label Elevated Label Flat Label Outlined Label Rounded Label Plain

Elevated Flat Outlined Rounded Plain

generic	> size	> minWidth	95px
style	> background	> bgColor__normal	#44ddbb
style	> shadow	> boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px
style	> label	> label__fontColor__normal	#FFFFFF

Linking input elements

You can establish a connection between the yLabel component and another component using a SourceLink Picker. In this example, the yLabel component is linked to a yInput component. Whenever the value in the yInput is changed, the connected yLabel component will update accordingly.

How to:

1. Open the Detail Panel and go to the **"Links"** tab to open the **"Label"** section.
2. Using the SourceLink Picker, select the previously created yInput.

That's it! You can now test this functionality by entering the preview mode and typing text in the yInput component. The yLabel component should update accordingly.

Label Input SourceLink

INFO

If you want to include text before or after the input value, you can add it in the **"Style"** > **"Label"** > **"Label"** property of the yLabel component. To display the input value, use {value} as a placeholder.

Label Template Result

Displaying table data

You can also establish a connection between the yLabel component and the yTable using a SourceLink Picker. In this example, the yLabel component is linked to specific column of the yTable. Whenever the selected row changes, the connected yLabel component will update accordingly.

How to:

1. Create a yTable in your application.
2. Open the Detail Panel of the yLabel and navigate to **"Links"** to unfold the **"Label"** group. Now, there are two options:
   i. Using the SourceLink Picker, select the previously created yTable and select a specific row in **"Collection/Buffer Property"**.
   ii. Using the SourceLink Picker, select the previously created yTable, leave the **"Collection/Buffer Property"** empty.
3. Change the **"Style"** > **"Label"** > **"Label"** Property. Here, you also have the option to include custom text in the message: "Hello {value}!".

Now, in the preview mode, when you click on a row in the yTable, the yLabel component will update accordingly.

Surname	Name	Subscribed
Max	Mustermann	true
Sabine	Mustermann	false
Jonas	Mustermann	true

Hello Sabine!

Edit this page

*Last updated by **Felix***

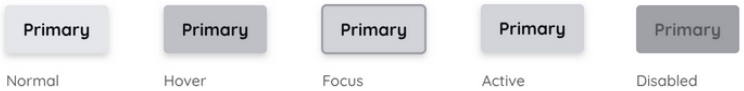
Previous

Next

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yTable



Description
Basic Structure
Properties
Usage
Variations
Blank
Scrollbehavior
Interactive Filtering
Toggle user actions
Pagination
Selection styling
Changing display types
Individual columns
Special Picker
Entity filter
Entity selector
Table dimensions

Description

The yTable component serves as a powerful tool primarily designed for displaying and interacting with an entity. It's an adaptable element that allows both static and dynamic content.

With the yTable you can display tables of any kind and data from any entity of your liking.

Some uses for the yTable are:

- Interactive Filtering
- Pagination

Basic Structure

In the following the basic structure of the yTable shall be explained. For the general structure of a yComponent please visit the YBase-components basic-structure.

The yTable as seen above contains a header, columns, rows and a footer.

Header

The header contains the title of the table and toggleable functions for adding rows or filtering columns. It is normally visible, but can also be toggled such as the elements inside the header. You can find further explanation at the TableHeader Property and TableHeader Title.

Columns

The columns consist of many sub components. Together the ColumnHeader, CounterColumn and UtilsColumn form the columns. All three of them are normally visible but can be made unseen. The CounterColumn contains a counter to show the actual index of a row. The UtilsColumn allows to have edit options for any row. The ColumnHeader renders the existing properties of an entity as columns.

Rows

The rows are the actual data entries of an entity and present this data visually. You can find further explanation at the Row Property.

Footer

Finally the TableFooter contains any options for pagination. It is normally visible, but can be toggled if needed. You can find further explanation at the TableFooter Property.

Properties

Through its various properties the **yTable** can be configured to suit your needs. The display below provides you with an overview of all the **yTable** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.

- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Table Links Events Columns

▶ META

▶ DISPLAY

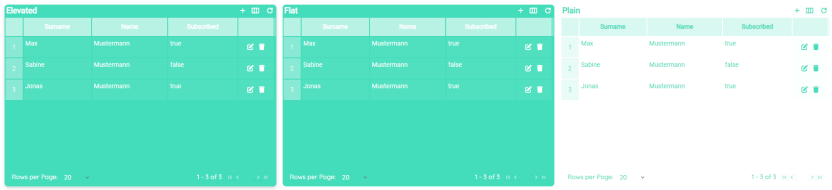
▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yTable in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations



Elevated Flat Plain

style	> tableHeader	> table-header__visible	false
style	> tableFooter	> table-footer__visible	false
style	> counterColumn	> show-counter-column	false
style	> utilsColumn	> show-utils-column	false

Blank

@Note insert image here

Blank

style	> tableHeader	> table-header__visible	false
style	> tableFooter	> table-footer__visible	false
style	> counterColumn	> show-counter-column	false
style	> utilsColumn	> show-utils-column	false

Scrollbehavior

For purposes of loading a certain amount of data, one needs to think about a balance between performance and usability. A huge amount of data can take time to fetch and to render.
Therefore we offer a possibility to address these problems and find a solution.

- **Basic**
  This causes the data to be loaded at once. Applicable for small amounts of data.
- **Infinite**
  At first only the visible data entries are rendered, whilst other entries only get loaded and rendered additionally, once the user is scrolling.

- **Virtual**
  During rapid scrolling, virtual scrolling is employed internally, accompanied by a simplified scroll animation. Actual data retrieval and rendering occur only during slower scrolling, ensuring that data is loaded only when the user can effectively engage with it.

Basic Infinite Virtual

style	> bodyCell	> body-cell__render-type	Basic

Interactive Filtering

In combination with the integrated SourceLink system, you can build an automatic and interactive filtering.

For instance you could filter the table based on a selection inside another yTable.

How to:

1. Change the module to @Note insert a hotlink here Entity-Editor.
2. Create two entities with names *"Order"* and *"Customer"*.
3. Inside the *"Order"* entity, create a number property: *"customerId"*.
4. Inside the *"Customer"* entity, create two string properties with the names *"surname"* and *"name"*.
5. Change the module to UI-Designer.
6. Create two yTable components in your application and connect each one to the entities you created before.
7. Select the created yTable which is connected to the *"Order"* table and proceed to the **"Links"** section within the Detail Panel.
8. Create a SourceLink on the other yTable on the **"select-row"** event and select **"_id"** as the Data-Property.
9. Insert following string into the **"template-filter"**: `customerId eq {value}`

When both entities contain valid data, the *"Order"* table gets filtered each time a row in the *"Customer"* table gets selected, filtering based in the *"customerId"*, showing only the orders the selected customer did.

Toggle user actions

1. **Header actions**
   Actions that are related to the table as a whole. This means that these actions affect the table itself.
2. **Row actions**
   These actions apply to single rows.

@Note insert image here

Header Row

style	> tableHeader	> add-record	false
style	> tableHeader	> visible-columns	false
style	> tableHeader	> refresh	false

Pagination

For instance the pagination is a complex construct for defining how many data entries are loaded per view, without the the necessity to change the table page.
For this purpose the table includes mechanisms to control this usecase.

How to:

1. Creating new pagination options
   i. Open the Detail Panel and navigate to the **"Style"** tab to open the **"TableFooter"** section.
   ii. Open the Optionlist Picker by clicking on **"rows-per-page-options"**.
   iii. This provides an overview of the existing options. Now create a new entry and set **"value"** and **"label"** to 75.
2. Setting an initial value:
   i. In the Detail Panel, navigate to the **"Style"** tab and open the **"TableFooter"** section.

ii. Set the **"rows-per-page"** to 75.

@Note insert image here

50 rows per page | 100 rows per page

style	> bodyCell	> body-cell__height	20px
style	> tableFooter	> rows-per-page	50

Selection styling

There are two different approaches of styling a selection inside the table, whic can also be combined.

- **Row**
- **Cell**

@Note insert image here

Row | Cell

style	> row	> row-selected__bg-color	#FF0000
style	> row	> row-selected__opacity	50%

Changing display types

The purpose of this feature presents it's necessity in displaying values of data entries in different variations.

While this might sound confusing for entries like e.g. names or numbers, in fact it's pretty useful for things like boolean values or images. For instance having an image url which holds a profile picture of every user of your guestlist might require rendering this image as a real image and not displaying the url of it.

> INFO
>
> When changing the display of a column, you always need to select one and make sure the changes you made apply for the data which is stored.
>
> E.g. while displaying a string "true" as a checkbox might work, displaying an url in the same way does not make sense at all.

@Note insert image here

Image | Boolean

columns	> generic	> display table	image
columns	> generic	> display edit dialog	text

Individual columns

In comparison to changing styles which apply to all cells, you can individualize each column to appear in a unique style.

This also includes the previous example Display of Cells and holds much more possibilities like changing sizing or coloration of special columns.

@Note insert image here

Sizing | Coloration

columns	> generic	> label	MyCustomColumn
columns	> generic	> body-cell__min-width	50px
columns	> column	> body-cell__font-size	22px
columns	> column	> body-cell__font-weight	500

Special Picker

Entity filter

With this picker you can adjust the yTable to liking by filtering the elements to be rendered.

@Note insert image here

Inside the header the rows per page can be defined. Here you can define a concrete number of rows or choose automatic determination which can be helpful.

The core of the picker shows a hypothetic presentation of the table itself. This interactive table allows you to choose all elements that should be displayed, simply by clicking them.

Entity selector

This picker allows you to select the current data source of your yTable and to select all properties that should be fetched.

@Note insert image here

First of all, a selection can be made of every entity in the current database. This implies that there is no selection of external data to this day.
After you chose an entity, it's properties can be filtered and a selection can be made. This affects which properties are being fetched from the database. Therefore only this data will be availlable until you change the selection.

> **INFO**
> This can't be compared to a classic filtering. You only get your selection of properties! So different from filtering, not all properties are being fetched and loaded.

Table dimensions

As the name implies you can set dimension related properties inside this picker.

@Note insert image here

The picker can be divided into three different areas:

- **Table**
  This manipulates the actual width and height of the table itself.
- **Cell**
  Here you can set the height for the cells inside the table.
- **Cell-Padding**
  This area offers the manipulation of the padding inside the cells

Docs

Tutorial

Community

Stack Overflow

Discord

Twitter

More

Blog

GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yButton

Description
Basic Structure
Properties
Usage
Variations
Label
Icon
Calling a custom JavaScript function
Hiding and showing Popups
Interacting with yeetClipboards

Description

The yButton is a clickable element with different optical styles, used to indicate its different states. The yButton can be used to trigger a user action. For example:

- Calling a custom JavaScript function
- Hiding and showing popups
- Interacting with yeetClipboards

Basic Structure

In the following the basic structure of the yButton shall be explained. For the general structure of a yComponent please visit the YBase-components basic-structure.

The yButton as seen above contains a label and a icon.

Label

The label contains the text inside the button. It is normally in the center of the component, but can also be displayed on the right or left site. By changing the inner padding of the button, the label gets moved too. You can find further explanation at the Label Property.

Icon

The icon can be used to show an icon inside the yButton. You can have a maximum of two icons in a yButton. The icon can be before or after the text and if you want to have only one yButton with one icon, then you have to delete the text in the Label. You can find further explanation at the Icon Property. An icon is structured as you can see below:

Properties

Through its various properties the **yButton** can be configured to suit your needs. The display below provides you with an overview of all the **yButton** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Button Events

► META

► DISPLAY

► SIZE

► PLACEMENT

Usage

In this section you'll find a collection of applicaton scenarios and examples that illustrate how to leverage the yButton in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Elevated Flat Outlined Rounded Plain

generic	> size	> minWidth	120px
style	> background	> bgColor__normal	#44d8db
style	> shadow	> boxShadow__normal	0 5px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px
style	> label	> label__fontColor__normal	#FFFFFF

Label

Uppercase Label

generic	> size	> min-width	150px
style	> label	> label__font-color	#FFFFFF
style	> label	> label__font-family	'Roboto'
style	> label	> label__font-size	20px
style	> label	> label__font-weight	'bold'
style	> label	> label__text-transform	'uppercase'
style	> label	> label	'Submit'

Icon

iconLeft iconBoth iconOnly

generic	> size	> minWidth	150px
style	> icon	> iconPosition	'iconLeft'
style	> icon	> icon__colorPrimary	#FFFFFF
style	> icon	> iconSize	16px
style	> icon	> iconLeft	'fas-fa-fish'

Calling a custom JavaScript function

The yButton component enables you to execute a custom JavaScript function each time it is clicked.

For instance, you could set up a function that creates a counter on the yButton, incrementing each time the yButton is pressed. Once the counter reaches the number 10, it automatically resets. The steps to achieve this are detailed below:

How to:

1. Open the Detail Panel and proceed to **"Events"**, unfold **"Input"** to modify **"EvtClicked"**.
2. Associate a custom JavaScript function to the yButton. This function will be invoked every time the yButton is clicked.
3. Include the provided JavaScript snippet below to implement a counter function on the yButton.

Code example:

You can add the following JavaScript code snippet to your application to implement a counter function on the yButton component.

With this script, each click on the yButton will increment the displayed counter until it reaches 10, upon which it will display a congratulatory message and reset. The yButton background color will also change depending on the counters current state.

Button EvtClicked

```
// function onButton_1_EvtClicked (apiObject, component, eventData) {

let buttonText = component.get("style", "label", "label");
// Convert string to number.
let counter = Number(buttonText);
// Not a number -> reset counter.
if (Number.isNaN(counter)) {
  counter = 0;
  component.set(
    "style",
    "background",
    "bgColor__normal",
    new apiObject.types.colorValue("#22aaffff")
  );
  component.set(
    "style",
    "background",
    "bgColor__hover",
    new apiObject.types.colorValue("#49B0FFFF")
  );
}
// Increment counter.
else {
  counter++;
}
// Count until ten.
if (counter == 10) {
  buttonText = "well done!";
  component.set(
    "style",
    "background",
    "bgColor__normal",
    new apiObject.types.colorValue("#00ff0000")
  );
  component.set(
    "style",
    "background",
    "bgColor__hover",
    new apiObject.types.colorValue("#00ff00FF")
  );
} else {
  buttonText = counter.toString();
}
// Log counter value in browser console.
console.log(counter);
// Set property value of button.
component.set("style", "label", "label", buttonText);

// }
```

Hiding and showing Popups

You can use the yButton to open and close popups.

How to:

1. Navigate to the popup you wish to control and open its Detail Panel.
2. Within these settings, proceed to the **"Links"** section and select **"Open"**.
3. In this area, use the SourceLink Picker to select the yButton you created earlier. As event choose **"click"**.

And that is it, you can now test this functionality by by entering the preview mode and clicking on the button to observe the popup's visibility change. Visit the popup section, if you want to know more about their behavior.

Interacting with yeetClipboards

You can use the yButton to control Clipboards. The yeetClipboard functionality provides an effective method for managing for example table entries. As example, you can set up a mechanism to delete a table entry by a simple button click.

How to:

1. Open the Detail Panel and proceed to the events tab and locate the **"Control"** section. Here, you'll need to enable the **"Commit-Delete"** option. This step empowers the yButton component with the capability to perform this functionality.
2. Create a new Clipboard and enable the **"Use Entity"** toggle.

3. Then you can click on **"Entity-Properties"** and edit the **"Entity"**. This Clipboard should be linked to the entity containing the table entry you wish to delete. This way you have to fill in the data, like in the picture below.

Database: https://rocket2.vectorsoft.de/api/DataStore/doku

Entity:

Entity	example
Sourcelink-1:	
Keys	
_id	0
Sourcelink-1:	

4. The _id under "Keys" should be the entry that you wish to delete.
5. Finally, go to the "Events", edit it and select the yButton as SourceLink-1 via the SourceLink Picker. Dont forget that the event has to be "commit-delete".

Events:

Select Row	
Sourcelink-1:	
Commit Create	
Sourcelink-1:	
Commit Read	
Sourcelink-1:	
Commit Update	
Sourcelink-1:	
Commit Delete	
Sourcelink-1:	y-page.ly-button.2.commit-delete

This will establish a button that, when clicked, will delete the specified entry from your table.
If you want to know more about this topic, visit the **Clipboard**.

Edit this page

Last updated by **felix**

Previous
« yTable

Next
yCheckbox »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yCheckbox

Description
Basic Structure
Properties
Usage
Variations
Custom Style Variations
Setting the Checkbox Value
Setting Error States
Integration with Other Components

Description

The yCheckbox is an essential user interface element that allows users to make binary choices or toggle between two states. It's commonly used in forms, settings, and wherever a selection is needed. Usecases can be:

- disabling features or other components
- requesting consent for terms and conditions
- adjusting settings for an app

Basic Structure

The basic structure of yCheckbox includes a checkbox icon and an optional label that describes its purpose.

@note Image

Label

The label is an optional text placed next to the checkbox, providing information about the choice it represents. It can be positioned on either side of the checkbox, depending on the layout requirements.

Box

The box represents the clickable area of the yCheckbox, showing an indicator when selected. It can be customized in terms of color, size, and border to match the application's design theme.

Properties

Through its various properties the **yCheckbox** can be configured to suit your needs. The display below provides you with an overview of all the **yCheckbox** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Checkbox Links Events

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yCheckbox in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Checkbox Elevated Checkbox Flat Checkbox Outlined Checkbox Rounded Checkbox Plain

Elevated Flat Outlined Rounded Plain

generic	> size	> minWidth	95px
generic	> size	> maxHeight	30px
style	> background	> bgColor__normal	#44ddbb
style	> shadow	> boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px
style	> label	> label__fontColor__normal	#FFFFFF
style	> box	> boxOff__bgColor	#FFFFFF
style	> box	> boxOff__borderColor	#FFFFFF
style	> box	> boxOn__bgColor	#FFFFFF
style	> box	> boxOn__borderColor	#FFFFFF
style	> box	> boxOn__iconColor	#44ddbb

Custom Style Variations

Configure your yCheckbox with unique styles to emphasize their importance or to better integrate with your application's design.

Checkbox Standard Checkbox Dark Mode

Standard Dark Mode

generic	> size	> minWidth	200px
style	> box	> boxOff__bgColor	#FFFFFF
style	> box	> boxOff__borderColor	#767676
style	> box	> boxOn__bgColor	#4A90E2
style	> box	> boxOn__borderColor	#4A90E2
style	> box	> boxOn__iconColor	#FFFFFF
style	> label	> label__fontColor	#333333
style	> label	> label	'Standard Checkbox'

Setting the Checkbox Value

The yCheckbox component's value can be dynamically assigned either programmatically via JavaScript or by establishing a link to another component's value—known as a SourceLink. This flexibility allows the yCheckbox to respond to user interactions or application states in real time.

Via JavaScript

To set the yCheckbox's value using JavaScript, access the value property within the **"style" > "thumb"** category. This method is ideal for changing the yCheckbox's state based on user interactions or other events within your application.

Here's how you can set the value in JavaScript:

EvtClicked

```
// function onCheckbox_1_EvtClicked (apiObject, component, eventData) {
var desiredValue = true; // or false
component.set("style", "thumb", "value", desiredValue);
// }
```

Simply replace desiredValue with the appropriate boolean or comparable value that corresponds to the yCheckbox's position.

Via SourceLink Picker

To link the yCheckbox's value with another component (e.g. another yCheckbox), use the SourceLink Picker. This feature binds the yCheckbox's state to another component's value, allowing for integrated component behavior.

How to:

1. Create another yCheckbox component in your application.
2. Select the created yCheckbox and open the Detail Panel.
3. Within these settings, proceed to the "Links" section and open "value".
4. Create a new SourceLink -> select the other yCheckbox you just created and choose "value" as an event.

Now each time you trigger the second yCheckbox, the first one will also be triggered. To make this both ways you have to repeat this process for the other yCheckbox as well.

Alternatively you can use the SourceLink Picker in the Toolbar as a Toolbar Extension.

Setting Error States

Use the yCheckbox to provide immediate feedback on form validations by setting an error state.

yButton:EvtChanged

```
// Example: Validate Checkbox State
// function onButton_1_EvtClicked (apiObject, component, eventData) {

leet checkboxComponent = apiObject.ui.getObject(checkboxId);
const isChecked =  checkboxComponent.get("style", "box", "value");

if (!isChecked) {
  // Set error state and message
  checkboxComponent.set("component", "error", "error", true);
  checkboxComponent.set("component", "error", "errorMessage", "This field is required.");
} else {
  // Clear error state
  checkboxComponent.set("component", "error", "error", false);
}
//}
```

Integration with Other Components

The yCheckbox can interact with other components, enabling or disabling functionalities based on its state. This behavior is essential for creating dynamic forms and interfaces.

Checkbox EvtClicked

```
// Example: Toggle visibility of another component based on checkbox state
// function onCheckbox_1_EvtClicked (apiObject, component, eventData) {

const isChecked = eventData.data;
let targetComponentId = 2;
let targetComponentType = "y-input";

const targetComponent = apiObject.ui.getObject(
  targetComponentId,
  targetComponentType
);

if (isChecked) {
  targetComponent.set("generic", "display", "visible", "visible");
} else {
  targetComponent.set("generic", "display", "visible", "hidden");
}

//}
```

Edit this page

Last updated by ***Wegener***

Previous
« yButton

Next
yExample »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yExample

Description
Basic Structure
Properties
Usage
Category
Category
Category
CATEGORY-THAT-HAS-A-SPECIAL-PB1-PLACEHOLDER
Category
Category
Category
Examples of use
Use Case 1
Use Case 2
Code examples

Description

The YExample is an element with different optical styles to indicate its different states. The YExample can be used to ...

Some uses for the YExample are:

- UseCase1
- UseCase2

Basic Structure

In the following the basic structure of the YExample shall be explained. For the general structure of a yComponent please visit the **YBase-components basic-structure**.

The YExample as seen above contains ...

Basic Element 1

The Element 1 can be used to... And contains...

Basic Element 2

The Element 2 can be used to... And contains...

Basic Element 3

The Element 3 can be used to... And contains...

Properties

Through its various properties the **YExample** can be configured to suit your needs. The display below provides you with an overview of all the **YExample** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Example Events

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the YExample in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Category

Category > Group

The easiest way to configure properties of the **GROUP-NAME-PLACEHOLDER** group, is to use the **PICKER-NAME-PLACEHOLDER**.

Important information about the **PROPERTY-NAME-PLACEHOLDER** property is...

> TIP
> If there is something important to say about a property then it should be added like this!

example configuration 1 example configuration 2 example configuration 3

category > group > property1 property-value1

Category

Category > GROUP-THAT-HAS-A-SPECIAL-PB2-PICKER-PLACEHOLDER

The **GROUP-THAT-HAS-A-SPECIAL-PB2-PICKER-PLACEHOLDER**-group can be either further configured in the **Detail Panel** or via the dedicated **GROUP-THAT-HAS-A-SPECIAL-PB2-PICKER-PLACEHOLDER** Picker that can be found in **Toolbar** by **ACTION-FOR-ACTIVATING-SPECIAL-PICKER-PLACEHOLDER**. There you can...

PROPERTY-FROM-PICKER-PLACEHOLDER

The "PROPERTY-FROM-PICKER-PLACEHOLDER"-property can be changed by "ACTION-TO-ADJUST-PROPERTY-PLACEHOLDER"...

CATEGORY-THAT-HAS-A-SPECIAL-PB1-PLACEHOLDER

The **CATEGORY-THAT-HAS-A-SPECIAL-PB1-PLACEHOLDER**-category's properties can be edited in a **special Toolbar** by **ACTION-FOR-ACTIVATING-SPECIAL-PB1-PLACEHOLDER**. There you can...

Inside the special Toolbar the following entries can be found:

- entry 1
- entry 2
- entry 3

Category > GROUP-THAT-IS-INSIDE-A-SPECIAL-PB1-PLACEHOLDER

The **GROUP-FROM-PICKER-PLACEHOLDER**-group's properties can be changed by **ACTION-TO-ADJUST-GROUP-PLACEHOLDER**.

PROPERTY-FROM-PICKER-PLACEHOLDER

The "PROPERTY-FROM-PICKER-PLACEHOLDER"-property can be changed by "ACTION-TO-ADJUST-PROPERTY-PLACEHOLDER"...

Category

GROUP-THAT-HAS-A-SPECIAL-PB1-PLACEHOLDER

The **GROUP-THAT-HAS-A-SPECIAL-PB1-PLACEHOLDER**-group's properties can be edited in a **special Toolbar** by **ACTION-FOR-ACTIVATING-SPECIAL-PB1-PLACEHOLDER**. There you can...

Inside the special Toolbar the following entries can be found:

- entry 1
- entry 2
- entry 3

PROPERTY-FROM-PICKER-PLACEHOLDER

The "PROPERTY-FROM-PICKER-PLACEHOLDER"-property can be changed by "ACTION-TO-ADJUST-PROPERTY-PLACEHOLDER"...

Category

Category > Group

Double clicking on Case1 enables the inPlace-edit for the element. Additionally the **special Toolbar Picker** becomes available.

Category

Category > Group

The group **GROUPNAME** only exists in **COMPONENTNAME**.It is used to configure the **STRUCTURE LINK** properties: **PROPERTIES**

Examples of use

Use Case 1

You can use the YExample to...

How to accomplish the use-case:

1. do this
2. do that
3. ...

Use Case 2

You can use the YExample to...

How to accomplish the use-case:

1. do this
2. do that
3. ...

Code examples

You can use the YExample to...

How to accomplish the use-case:

1. do this
2. do that
3. ...

Code Example

This action has four parameters and returns a new string built from those parameters. In this example the user can submit the arguments via input fields and start the action by clicking on the "Send gift" button.

yButton script example

```
let action = api.actions.create("SendGift");
if (action !== undefined) {
  // Get input field objects by their IDs
  let cstNameObject = api.ui.getObject(12); // inp_name
  let cstAgeObject = api.ui.getObject(13); // inp_age
  let cstGiftObject = api.ui.getObject(15); // inp_gift
  let cstMsgObject = api.ui.getObject(16); // inp_text
  // Create JSON with arguments for RPC and write values of input fields to JSON object
  let jsonArgument = {
    CustName: cstNameObject.get("style", "content", "valueString"),
    CustAge: cstAgeObject.get("style", "content", "valueNumber"),
    CustText: cstMsgObject.get("style", "content", "valueString"),
    CustGift: cstGiftObject.get("style", "content", "valueString"),
  };
  // Keep RPC result stored on server for 300 seconds
  let tiSeconds = 300;
  action.execute(jsonArgument).then(
    (success) => {
      action.awaitResult().then(
        (success) => {
          console.log(success);
          api.ui.getObject(20).set("style", "content", "value", success.value);
        },
        (error) => {
          alert("error awaitResult");
        }
      );
    },
    (error) => {
      alert("error execute action");
    }
  );
}
```

Edit this page

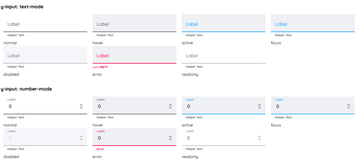
Last updated by **Felix**

Previous
« yCheckbox

Next
yInput »

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yInput



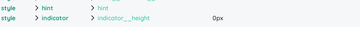
Description

The Yinput is an element that allows user input. This Yinput can be furhter used for example for:

- Asking user information like a username with custom validation
- Other inputs with masks or inbuild validation

Basic Structure

In the following the basic structure of the Yinput shall be explained. For the general structure of a yComponent please visit the **YBase-components basic-structure**.



The Yinput as seen above contains a label and a helper text.

Label

The label contains the text inside the button. It is on the right side of the component. By changing the inner padding of the button, the label gets moved too. You can find further explanation at the **Label Property**.

Helper Text

The Helper Text is similar to the label and displays a hint related to the input that will be expected. In certain situations the helper text will be replaced by a warning or error text, which can either be set through the automatic validation or custom scripts.

Properties

Through its various properties the **YInput** can be configured to suit your needs. The display below provides you with an overview of all the **YInput** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Input Links Events

▸ META

▸ DISPLAY

▸ SIZE

▸ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the Yinput in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Elevated Flat Outlined Rounded Plain

generic	size	minWidth	200px
generic	size	maxHeight	44px
style	background	bgColor__normal	#44ddbb
style	shadow	boxShadow__normal	0 5px 6px 0 rgba(16, 18, 24, 0.2)
style	border	borderRadius	8px
style	label	label__fontColor__normal	#FFFFFF
style	hint	hint	
style	indicator	indicator__height	0px

Special Variations

Beyond standard configurations, the Yinput allows for advanced customization. Tailor the component for specific needs like user feedback forms with character limits or code input areas with monospace font styling.

Numeric Context Textual Input No Label, Placeholder Prefix & Suffix Prefix Icon

generic	size	minWidth	200px
generic	size	maxHeight	44px
style	label	label	'Enter amount'
style	placeholder	placeholder	'e.g., 100'
style	content	value	'100'
style	hint	hint	'Numeric values only'
style	content	content__fontAlign	'right'
style	background	bgColor__normal	#ECEFF1
style	border	borderStyleTop__normal	solid
style	border	borderStyleRight__normal	solid
style	border	borderStyleBottom__normal	solid
style	border	borderStyleLeft__normal	solid
style	border	borderWidthTop__normal	2px
style	border	borderWidthRight__normal	2px
style	border	borderWidthBottom__normal	2px
style	border	borderWidthLeft__normal	2px
style	border	border-colorTop__normal	#CFD8DC
style	border	border-colorRight__normal	#CFD8DC
style	border	border-colorBottom__normal	#CFD8DC
style	border	border-colorLeft__normal	#CFD8DC
style	content	type	'number'

Mask and Validation

In the component > validation group you will find the mask and validate properties. Mask is the value to be used for the inbuilt masks and validate an option to add custom validation to the input.

Incorporating Maska into our input component allows for the creation of input masks using strings, enabling the enforcement of specific formats for data entry such as telephone numbers and birthdays. To define a mask, simply provide a string that represents the desired format. For example, a mask for a german telephone number could be defined as `'+49 (###) ###-####'`, where **#** represents a digit placeholder. This ensures that users input their telephone numbers in a consistent and predefined format, enhancing data uniformity and reducing entry errors.

Additionally, our component features a validation picker accessible via a plus icon, allowing for the addition of custom validation conditions. Within this picker, you can specify a comparator (e.g., equals, not equal, greater than, etc.), a value to compare against, and a custom error message to display in case the validation fails. This error message is displayed in place of the usual hint, providing immediate and clear feedback to the user regarding the nature of the input error. This functionality not only enhances the robustness of data validation but also improves the user experience by guiding users towards correct data entry.

In the following you can see an example of the validation group.

Debounce

The debounce function which can be found in the component > misc group is a powerful feature designed to enhance the performance and user experience of applications by controlling how frequently a function can be executed. This is particularly useful in scenarios where you have events that trigger rapidly or continuously, such as during typing in a search field or resizing a window. By implementing a debounce function, you can set a delay (in milliseconds) that determines how long the system should wait after the last event trigger before executing the function. If a new event occurs before the delay has elapsed, the timer resets. This effectively reduces the number of times the function is called, minimizing unnecessary computations and API calls.

How to:

1. Open the Detail Panel and proceed to **"Component"**, unfold **"Misc"** to modify **"Debounce"**.

For example, when applied to an input field for live search, debounce can ensure that the search query is sent to the server only after the user has stopped typing for a specified duration, rather than sending a request with every keystroke. This not only reduces the load on the server but also provides a smoother experience for the user.

In the documentation below, you'll find a code example that can be used together with the debounce function.

Custom Validation

Input EvtChanged

```
// function onInput_1_EvtChanged (apiObject, component, eventData) {

const userName = eventData.data.trim();

if (userName.length < 3) {
  alert("The username must be at least 3 characters long.");
  return false;
} else if (/[^a-zA-Z0-9]/.test(userName)) {
  alert("The username can only contain alphanumeric characters.");
  return false;
}

console.log("Username is valid.");
return true;
//}
```

Special Toolbars

Default Toolbar with SourceLink Picker

The label and the main body of the Yinput component utilize the default Toolbar, as outlined on the Toolbar documentation page. This includes all standard settings for modifying appearance, behavior, and content properties. Additionally, this Toolbar is enhanced with a SourceLink Picker feature, offering the ability to establish dynamic links between the Yinput and other components or data sources within your application. This functionality is crucial for creating responsive, data-driven interfaces.

Specialized Detail Panel for Hint and Helper Text

Both the hint and the helper text elements of the Yinput component are associated with a specialized version of Detail Panel. This unique Toolbar simplifies the customization process, focusing on two key aspects:

- **State Picker**
  Allows you to navigate through various states of the Yinput component, such as hover, active, focused, etc. This feature is instrumental in designing a user interface that reacts intuitively to user interactions, providing visual feedback that enhances the overall user experience. @note Link needed
- **Text-Font Picker**
  As described on the Toolbar documentation page, the Text-Font Picker enables detailed customization of the text appearance within the hint and helper text elements. This includes adjustments to font size, weight, style, and color, ensuring that these textual elements are both aesthetically pleasing and aligned with the application's design standards. @note Link needed

Special Picker

SourceLink Picker

There is a special picker for the Yinput in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of 2 or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Description
Basic Structure
Properties
Usage
Variations
Special Variations
Mask and Validation
Debounce
Custom Validation
Special Toolbars
Default Toolbar with SourceLink Picker
Specialized Detail Panel for Hint and Helper Text
Special Picker
SourceLink Picker

Edit this page

Last updated by **kregener**

Previous
« yExample

Next
yQRCode »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yQRCode

Description

The yQRCode is a scannable code typically used for storing URLs or other information for reading by the camera on a smartphone. The yQRCode is versatile and can be used for:

- Directing users to websites quickly
- Simplifying the check-in process at events
- Sharing Wi-Fi credentials effortlessly

Basic Structure

The basic structure of the yQRCode includes customizable options such as the QR code's color, size, and the actual data encoded within it. For the general structure of a yComponent, refer to the YBase-components basic-structure.

BasicStructure

Properties

Through its various properties the **yQRCode** can be configured to suit your needs. The display below provides you with an overview of all the **yQRCode** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Links

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yQRCode in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

QRCode Elevated QRCode Flat QRCode Outlined QRCode Rounded QRCode Plain

Elevated Flat Outlined Rounded Plain

generic	> size	> minWidth	120px
style	> background	> bgColor__normal	#44ddbb
style	> shadow	> boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px
style	> code	> code__color	#FFFFFF

Style Variations

QRCode White QRCode Red

Red QR Code White QR Code

style	> background	> bgColor	#FF0000
style	> code	> code__color	#FFFFFF

Dynamic Content

The yQRCode also supports dynamic content, where the data encoded in the QR code can be updated in real-time based on user interactions or other events in the application.

Via JavaScript

To set the yQRCode's value using JavaScript, access the value property within the **"style" > "code"** category. This method is ideal for changing the yQRCode's state based on user interactions or other events within your application.

How to:

Button EvtClicked

```
// function onButton_1_EvtClicked (apiObject, component, eventData) {

let desiredString = "https://example.com";
let QRComponent = apiObject.ui.getObject(2, "y-qr-code");

QRComponent.set("style", "code", "code__value", desiredString);

//}
```

Simply replace desiredString with the appropriate string or URL to a website that you would like to encrypt in the QRCode.

In the following more advance code example a yInput named "yInputUrl" is needed and a yQRCode named "yQRCode". The content of the following code example can be run from any function that is triggered inside the page that contains those two components.

Input EvtChanged

```
// function onInput_1_EvtChanged (apiObject, component, eventData) {
// Retrieve the user input value from an input component
let urlValue = eventData.data;
const qrComponent = apiObject.ui.getObject(2, "y-qr-code");

// Check if the URL is not empty
if (urlValue.trim() !== "") {
  // Update the QR Code component's content
  qrComponent.set("style", "code", "code__value", urlValue);
  console.log("QR Code updated with URL:", urlValue);
} else {
  // Handle empty input, potentially showing an error or alert
  console.error("URL input is empty. QR Code cannot be updated.");
}
//}
```

Via SourceLink Picker

To link the yQRCode's value with another component (e.g. a YInput), use the SourceLink Picker. This feature binds the yQRCode's state to another component's value, allowing for integrated component behavior.

How to:

1. Create a yQRCode component in your application.
2. Create a YInput component in your application.
3. Select the created yQRCode and open the **Detail Panel**.
4. Within these settings, proceed to the "Links" section and open "QrCode".
5. Create a new sourceLink -> select the YInput you just created and choose **input-string** as an event.

Now each time you insert a url into the input, the image will render the picture behind this url. You can implement a validation in the YInput to make sure the string is in a format that you want, e.g. a valid URL.

Alternatively you can use the SourceLink Picker in the Toolbar as a Toolbar Extension.

Special Picker

SourceLink Picker

There is a special picker for the yQRCode in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of 2 or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Description
Basic Structure
Properties
Usage
Variations
Style Variations
Dynamic Content
Special Picker
SourceLink Picker

Edit this page

*Last updated by **Felix***

Previous
« yInput

Next
yRadio »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yRadio

yradio

Description

The yRadio is a clickable element with different optical styles, used to indicate its different states. The yRadio can be used to toggle a state or mirror a user selection where only one selection is possable. For example:

- Choosing a shipment method
- Toggling modes for the view of your page

Basic Structure

In the following the basic structure of the yRadio shall be explained. For the general structure of a yComponent please visit the **YBase-components basic-structure**.

yradio

The yRadio as seen above contains a label and the radio.

Label

The label contains the text inside the button. It is on the right side of the component. By changing the inner padding of the button, the label gets moved too. You can find further explanation at the **Label Property**.

Radio

A radio button, often referred to as a radio bubble, is a graphical user interface element that allows the user to choose only one option from a set of predefined options. It appears as a small circle that, when selected, is filled or marked with a dot or check inside. Radio buttons are typically grouped together to form a radio group, ensuring that selecting one button will deselect any other previously selected button within the same group. This makes them ideal for presenting exclusive choices where only one selection is allowed, such as choosing a shipping option, selecting a gender, or picking a payment method.

Properties

Through its various properties the **yRadio** can be configured to suit your needs. The display below provides you with an overview of all the **yRadio** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accessible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Radio Links Events

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yRadio in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

yRadio Elevated yRadio Flat yRadio Outlined yRadio Rounded yRadio Plain

Elevated Flat Outlined Rounded Plain

generic	size	minWidth	95px
generic	size	maxHeight	30px
style	background	bgColor__normal	#44dbb
style	shadow	boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	border	borderRadius	8px
style	label	label__fontColor__normal	#FFFFFF
style	radio	radioOff__bgColor	transparent
style	radio	radioOff__borderColor	#FFFFFF
style	radio	radioOn__bgColor	transparent
style	radio	radioOn__borderColor	#FFFFFF
style	radio	radioOn__iconColor	#FFFFFF

Radio Stylings

yRadio Minimalist yRadio Bold yRadio Luminous

Minimalist Bold Luminous

generic	size	maxWidth	120px
style	radio	radio__circleAnimationColorOpacity	0

Label Variations

yRadio Label Left yRadio Label Right yRadio Label Both

Label Left Label Right Label Both

generic	size	minWidth	'auto'
generic	size	maxWidth	none
style	label	leftLabel	'left'

Dynamic Value Assignment

The yRadio component is designed to facilitate dynamic user interactions within your application. It achieves this through two primary properties found within the **"style" > "radio"** category, value and comparisonVal.

Value Property

The value property is intended to be set externally, either by user action or programmatically via JavaScript. This property represents the current state or selection of the yRadio component.

Via Javascript

Button EvtClicked

```
// function onButton_1_EvtClicked (apiObject, component, eventData)
const shippingMethod = "LKW";
const radioComponent = apiObject.ui.getObject(1, "y-radio");
radioComponent.set("style", "radio", "value", shippingMethod);
//}
```

In this snippet, yourDesiredValue is the actual value that you want the yRadio to hold, which will reflect the user's selection. Alternatively, yRadio can obtain its value through a SourceLink. This method establishes a connection between yRadio and another component. The value received through the SourceLink is then compared against the comparisonVal to determine the radio button's state. The value can be coupled to a Clipboard value via the SourceLink Picker.

Via SourceLink

1. Create a yRadio component in your application.
2. Create a yInput component in your application.
3. Select the created yRadio and open the **Detail Panel**.
4. Within these settings, proceed to the "Links" section and open "Value".
5. Create a new sourceLink -> select the yInput you just created and choose **input-string** as an event.

In this example you should find a way to demand either by mask/validation or javascript to make sure the input-string is represented as a radio-value.

ComparisonVal Property

The comparisonVal serves as a unique identifier for a yRadio component within a group of similar radio buttons. To create a set of mutually exclusive selections — where choosing one option deselects the others — multiple yRadio components can share the same comparisonVal.

For instance, if you have a group of radio buttons for selecting a shipping method, each radio button's comparisonVal might be set to "shippingMethod". The one whose value matches the comparisonVal will be marked as selected:

yButtonEvtClicked

```
// function onButton_1_EvtClicked (apiObject, component, eventData)
const shippingMethod = "SKM";
const radioComponent = apiObject.ui.getObject(1, "y-radio");
radioComponent.set("style", "radio", "comparisonVal", shippingMethod);
//}
```

The ComparisonVal should only seldomly be changed via javascript.

Coupling and Selection State

When you have multiple radio buttons with the same comparisonVal, they are effectively coupled. The selection state (true or false) of these yRadio components is determined by comparing their value with their comparisonVal. When the value matches the comparisonVal, that particular radio button is selected (its state is true). If they don't match, the radio button remains unselected (its state is false).

This mechanism ensures that within a set of yRadio components with the same purpose, only the one with a matching value will be selected, enforcing the rule that only one option can be chosen at a time in a radio group.

By leveraging the value and comparisonVal properties, yRadio components can be effectively used to create a responsive and intuitive selection interface within your application.

Real Life Use Cases

Payment for a shop

This example provides the code to get the selected Payment-Method from radio buttons without the usage of a sourceLink for an external radio value. When a button is clicked the selected payment method will be fetched.

Button EvtClicked

```
// function onButton_1_EvtClicked (apiObject, component, eventData)
// Assume we have 3 radio buttons for payment methods: "Credit Card", "PayPal", "Bank Transfer"
// Each radio button has a unique ID

let selectedMethod;
const paymentMethods = [
  {
    name: "Credit Card",
    radioId: 4,
  },
  {
    name: "PayPal",
    radioId: 5,
  },
  {
    name: "Bank Transfer",
    radioId: 6,
  },
];

paymentMethods.forEach((method) => {
  const radioButton = apiObject.ui.getObject(method.radioId, "y-radio");
  const selectedValue = radioButton.get("style", "radio", "value");
  const comparisonValue = radioButton.get("style", "radio", "comparisonValue");
  const isSelected = selectedValue == comparisonValue;

  if (isSelected) {
    selectedMethod = method.name;
    console.log(`Selected payment method: ${method.name}`);
  }
});

if (!selectedMethod) {
  alert("Please select a payment method.");
  return false;
}

return true;
//}
```

Displaymodes

In this example three radios with the keys depicted in displayModes variables are needed. The following function has to be included in each of those radios as the 'EvtClicked'-Function.

Radio EvtClicked

```
// function onRadio_1_EvtClicked (apiObject, component, eventData)
// Function to toggle UI elements based on the selected display mode using YRadio
const displayModes = {
  BasicView: ["basicElement1", "basicElement2"], // List of elements to be shown in Basic View
  AdvancedViews: ["advancedElement1", "advancedElement2"], // List of elements for Advanced View
  ExpertView: ["expertElement1", "expertElement2"], // List of elements for Expert View
};

// Assume the radio buttons have IDs that match the display modes
const selectedMode = component.get("style", "radio", "value");

// Hide all elements first
Object.values(displayModes)
  .flat()
  .forEach(elementId => {
    const element = apiObject.ui.getObject(elementId);
    element.set("generic", "display", "visible", "none");
  });

// Show elements for the selected mode
if (displayModes[selectedMode]) {
  displayModes[selectedMode].forEach(elementId => {
    const element = apiObject.ui.getObject(elementId);
    element.set("generic", "display", "visible", "block");
  });
  console.log(`Display mode set to: ${selectedMode}`);
} else {
  console.error("Selected mode is not recognized:", selectedMode);
}
//}
```

Special Picker

SourceLink Picker

There is a special picker for the yRadio in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of Z or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Description
Basic Structure
Properties
Usage
Variations
Radio Stylings
Label Variations
Dynamic Value Assignment
Real Life Use Cases
Special Picker
SourceLink Picker

Edit this page

Last updated by **Wagener**

Previous
« yQRCode

Next
ySelectionPicker »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
- yPage
- yPopup
- yContainer
- yImage
- yLinkButton
- yLabel
- yTable
- yButton
- yCheckbox
- yExample
- yInput
- yQRCode
- yRadio
- ySelectionPicker
- yTextarea
- yTextbox
- yToggle
- yToggleButton

Essentials
yeet Platform
Advanced

ySelectionPicker

Description
- Basic Structure

Properties
Usage
- Variations
- Special Variations
- Display Modes
- SourceLinks
- Validation
- OptionList
- Payment Selection
- User Role Selection

Special Picker
- SourceLink Picker

Description

The *ySelectionPicker* is a UI component designed for creating selectable lists in web applications. It provides a user-friendly interface for selecting one or more items from a given set of options. The ySelectionPicker can for example be used for:

- Choosing a shipment method
- Choosing custom user roles

Basic Structure

In the following the basic structure of the ySelectionPicker shall be explained. For the general structure of a yComponent please visit the

@note image

The ySelectionPicker as seen above contains a label and a helper text.

Label

The label contains the text inside the ySelectionPicker. It is on the left side of the component. It will get moved to the top, when a selection is document. The label can be clicked to trigger a Selection to make the selection list visible. You can find further explanation at the **Label Property**.

Helper Text

The Helper Text is similar to the label and displays a hint related to the input that will be expected. In certain situations the helper text will be replaced by a warning or error text, which can either be set through the automatic validation or custom scripts.

Spinner Icon

The spinner icon can also be used to make a selection and make the selection list visible. It is on the right side of the ySelectionPicker.

Properties

Through its various properties the **ySelectionPicker** can be configured to suit your needs. The display below provides you with an overview of all the **ySelectionPicker** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style SelectionPicker Links Events

- ▶ META:read more
- ▶ DISPLAY:read more
- ▶ SIZE:read more
- ▶ PLACEMENT:read more

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the ySelectionPicker in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Elevated | Flat | Outlined | Rounded | Plain

Elevated Flat Outlined Rounded Plain

generic	size	minWidth	200px
generic	size	maxHeight	44px
style	background	bg-color__normal	#44c8bb
style	shadow	box-shadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	border	border-radius	8px
style	label	label__font-color__normal	#FFFFFF
style	hint	hint	
style	indicator	indicator__height	0px

Special Variations

Clearable MultiSelect With Search Icon Only No Label Icons And Label

component	state	clearable	true
style	background	bg-color__normal	#FFFFFF
style	border	border-color__normal	#E0E0E0
style	label	label__font-color__normal	#22a0ff
links	content	sourceLinkContent	'/api/options-list'

Options can include clearable selections, multi-select capabilities, and a search function. You can also have configurations with no label or hint text for a minimalistic approach.

Display Modes

The picker supports different modes for displaying options:

- **showOnlyIcons**
  Only icons are visible for each item.
- **iconsAndLabel**
  Both icons and labels are shown.
- **labelOnly**
  Only the text label of each item is displayed.
- **itemSeparator**
  Defines a visual separator between items.
- **activeItem**
  Highlights the currently selected item.

SourceLinks

SourceLinks for label, hint, and content can be configured to fetch data dynamically or link to different parts of the application. For more Information see the **SourceLinks** section.

Validation

The ySelectionPicker includes an error state that can be used to display validation messages. The validation has to be done manually via javascript.

EvtListSelect

```
// enabled async function
// async function onSelectionPicker_1_EvtListSelect (apiObject, component, eventData) {

const selectedProduct = eventData.data;
const isOnStock = testIfProductIsInStock(eventData.data); //own global function

if (!isOnStock) {
  // Display validation error message
  component.set("component", "state", "error", true);
  component.set(
    "component",
    "error",
    "errorMessage",
    "Please enter a valid search term."
  );
} else {
  // Clear error message
  component.set("component", "state", "error", false);
}
//}
```

OptionList

You can set the list of selectable options using the **"optionList"** property which can be found in style > list group. There you can add entries and give them a unique value, a label and a tooltip. Read more about the OptionList here.

How to:

1. Open Detail Panel: Use the Toolbar to access detailed settings.
2. Navigate to **"Style"** Category: Open the category to adjust component-specific settings.
3. Access **"List"** Group: Here you can find the options list.
4. Edit Options List: Click on the **"optionList"** to open the configuration window where you can add new entries (value, label, tooltip) or modify existing ones.

Payment Selection

SelectionPicker EvtListSelect

```
// function onSelectionPicker_1_EvtListSelect (apiObject, component, eventData)
const selectedValue = component.get("style", "content", "value");
const proceedButton = apiObject.ui.getObject("proceedButtonId"); // Replace with actual ID
const validPaymentMethods = ["creditCard", "paypal", "bankTransfer"];

if (validPaymentMethods.includes(selectedValue)) {
  // Enable the proceed button if a valid payment method is selected
  proceedButton.set("component", "state", "disabled", false);
  console.log("Valid payment method selected:", selectedValue);
} else {
  // Keep the proceed button disabled if the selection is not valid
  proceedButton.set("component", "state", "disabled", true);
  console.error("Invalid payment method selected:", selectedValue);
}
// }
```

User Role Selection

SelectionPicker EvtListSelect

```
// function onSelectionPicker_1_EvtListSelect (apiObject, component, eventData)
const selectedValue = component.get("style", "content", "value");
const additionalOptionsPanel = apiObject.ui.getObject(
  "additionalOptionsPanelId"
); // Replace with actual ID
const rolesWithAdditionalOptions = ["administrator", "editor"];

if (rolesWithAdditionalOptions.includes(selectedValue)) {
  // Show additional options if the selected role has extra permissions
  additionalOptionsPanel.set("generic", "display", "visible", "visible");
  console.log("Additional options displayed for role:", selectedValue);

  // You might want to fetch and display these additional options dynamically
  fetchAndDisplayAdditionalOptions(selectedValue);
} else {
  // Hide additional options if the selected role does not have extra permissions
  additionalOptionsPanel.set("generic", "display", "visible", "hidden");
  console.log("No additional options for role:", selectedValue);
}
// A mock function to fetch additional options based on the selected role
function fetchAndDisplayAdditionalOptions(role) {
  // Fetching logic here...
  console.log("Fetching additional options for role:", role);
  // Display logic here...
}
// }
```

Special Picker

SourceLink Picker

There is a special picker for the ySelectionPicker in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of 2 or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Edit this page

Last updated by **Wegener**

Previous
« yRadio

Next
yTextarea »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yTextarea

Description
Basic Structure
Properties
Usage
Variations
Special Variations
Custom Validation and Mask Usage
Handling Large Text with Auto-Expand
Special Picker
SourceLink Picker
Special Toolbars
Default Toolbar with SourceLink Picker
Specialized Toolbar for Hint and Helper Text

Description

The yTextarea component is designed for multi-line text input, offering extensive functionalities for text manipulation and display. It's particularly useful for cases where users need to input longer forms of text, such as comments, descriptions, or any other form of extended input.

Basic Structure

The yTextarea component is comprised of several parts, including the label, content area, hint text, and optional icons for additional functionalities. Here's a closer look at its basic structure:

Label

Provides a title or description for the yTextarea, indicating what type of input is expected.

Content Area

The main area where users can input their text. It can be resized to accommodate more content.

Hint Text

Offers additional guidance or examples to help users understand what they should input.

Properties

Through its various properties the **yTextarea** can be configured to suit your needs. The display below provides you with an overview of all the **yTextarea** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Textarea Links Events

▶ META

▶ DISPLAY

▶ SIZE

▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yTextarea in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Textarea Elevated Textarea Flat Textarea Outlined Textarea Rounded Textarea Plain

Elevated Flat Outlined Rounded Plain

generic	size	minWidth	200px
generic	size	maxWidth	200px
generic	size	maxHeight	100px
style	background	bgColor__normal	#44ddbb
style	shadow	boxShadow__normal	0.5px 6px 0 rgba(16, 18, 24, 0.2)
style	border	borderRadius	8px
style	label	label__fontColor__normal	#FFFFFF
style	hint	hint	
style	indicator	indicator__height	0px

Special Variations

Beyond standard configurations, the yTextarea allows for advanced customization. Tailor the component for specific needs like user feedback forms with character limits or code input areas with monospace font styling.

Textarea Blank Textarea Colorized

No Label, Placeholder Colorized

generic	size	minWidth	200px
generic	size	maxWidth	200px
generic	size	maxHeight	100px
style	label	label	
style	hint	hint	
style	placeholder	placeholder	
style	content	value	
style	indicator	indicator__height	0px
style	border	borderStyleTop__normal	solid
style	border	borderStyleRight__normal	solid
style	border	borderStyleBottom__normal	solid
style	border	borderStyleLeft__normal	solid
style	border	borderWidthTop__normal	2px
style	border	borderWidthRight__normal	2px
style	border	borderWidthBottom__normal	2px
style	border	borderWidthLeft__normal	2px
style	border	borderColorTop__normal	#000000
style	border	borderColorRight__normal	#000000
style	border	borderColorBottom__normal	#000000
style	border	borderColorLeft__normal	#000000

Compared to the yInput component here it is not possible to request only number input. This kind of validation has to be done manually via javascript.

Custom Validation and Mask Usage

Because there is no built in validation and mask features like in the input component we have to create this functionality by ourselves. Via custom javascript create a yTextarea with custom validation for email input and apply a mask for phone number formatting. Implement dynamic validation feedback to guide user input.

Textarea EvtChanged

```
// function onTextarea_1_EvtChanged (apiObject, component, eventData) {
// Assuming a yTextarea component with ID "emailTextarea" for email validation
const emailInput = component.get("style", "content", "value");
const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
if (!emailRegex.test(emailInput)) {
  // Display validation error message
  component.set(
    "component",
    "error",
    "errorMessage",
    "Please enter a valid email address."
  );
} else {
  // Clear error message
  component.set("component", "error", "errorMessage", "");
}
// }
```

Handling Large Text with Auto-Expand

Configure yTextarea to automatically expand based on content, enhancing user experience for inputs like user-generated stories or extensive comments.

Textarea EvtChanged

```
// function onTextarea_1_EvtChanged (apiObject, component, eventData) {
// Retrieve the current text from the textarea
const currentText = component.get("content", "value");
// Estimate the height required based on the text length
// This is a simplified example. Consider line breaks and other factors in a real implementation.
const lines = currentText.split("\n").length;
const lineHeight = 20; // Assuming each line requires 20 pixels
const padding = 20; // Top and bottom padding
const minHeight = 100; // Minimum height of the textarea
const newHeight = Math.max(minHeight, lines * lineHeight + padding);

// Adjust the height of the textarea based on the content
component.set("style", "size", "minHeight",  ${newHeight}px );
// }
```

These examples showcase the flexibility of yTextarea, catering to a wide range of use cases from simple text inputs to complex data entry forms with validation and formatting.

Special Picker

SourceLink Picker

There is a special picker for the yTextarea in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of 2 or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Special Toolbars

Default Toolbar with SourceLink Picker

The label and the main body of the yTextarea component utilize the default Toolbar, as outlined on the Toolbar documentation page. This includes all standard settings for modifying appearance, behavior, and content properties. Additionally, this Toolbar is enhanced with a SourceLink Picker feature, offering the ability to establish dynamic links between the yTextarea and other components or data sources within your application. This functionality is crucial for creating responsive, data-driven interfaces.

Specialized Toolbar for Hint and Helper Text

Both the hint and the helper text elements of the yTextarea component are associated with a specialized version of the Toolbar. This unique Toolbar simplifies the customization process, focusing on two key aspects:

- **State Picker**
  Allows you to navigate through various states of the yTextarea component, such as hover, active, focused, etc. This feature is instrumental in designing a user interface that reacts intuitively to user interactions, providing visual feedback that enhances the overall user experience.
- **Text-Font Picker**
  As described on the Toolbar documentation page, the Text-Font Picker enables detailed customization of the text appearance within the hint and helper text elements. This includes adjustments to font size, weight, style, and color, ensuring that these textual elements are both aesthetically pleasing and aligned with the application's design standards.

Edit this page

Last updated by **Wegener**

Previous
« ySelectionPicker

Next
yTextbox »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yTextbox

Description
Basic Structure
Properties
Usage
Variations
Special Variations
Special Picker
Rich text editor

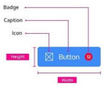
Description

The yTextbox component serves as a powerful tool primarily designed for displaying longer text information within your application. It's an element that allows only static content. The yTextbox basically can be used to store text information that goes beyond the context of a simple label.

Basic Structure

In the following the basic structure of the yTextbox shall be explained. For the general structure of a yComponent please visit the

YBase-components basic-structure.



The yTextbox as seen above contains nothing but text as the content.

Content

The content contains the actual text itself. It can be seen as an equivalent to the label of the

yLabel. You can find further explanation at the [Content Property](#style--content).

Properties

Through its various properties the **yTextbox** can be configured to suit your needs. The display below provides you with an overview of all the **yTextbox** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic Style Events

- ▶ META
- ▶ DISPLAY
- ▶ SIZE
- ▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yTextbox in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

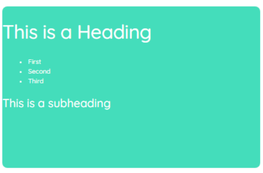
Textbox Elevated Textbox Flat Textbox Outlined Textbox Rounded Textbox Plain

Elevated Flat Outlined Rounded Plain

generic	size	minWidth	200px
generic	size	maxWidth	200px
generic	size	height	100px
style	background	bgColor__normal	#44ddbb
style	shadow	boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	border	borderRadius	8px
style	label	label__fontColor__normal	#FFFFFF
style	content	value	' This is an example text: <ol><li> Elevated </li></ol>'

Special Variations

Textbox Tags Textbox Links



Using tags Styling links Different headings

generic	size	minWidth	200px
generic	size	maxWidth	200px
generic	size	height	100px
style	content	value	Hello world! <a href='www.vectorsoft.de'>Click here :-)<a>

Special Picker

Rich text editor

This special editor helps creating a content of your liking. The principle can be compared to a common text editor.

@Note hier Bild einfügen

The editor can be activated by double clicking the textbox in the @Note insert Link **Designer**. Now the textbox turns from simply displaying the content to an editor mode. In this mode you can simply create, edit and delete content.

The Toolbar hereby offers several options:

1. **Undo/Redo**
2. **Changing fonts**
3. **Changing font styles**
4. **Creating link**
5. **Inserting ordered and unordered lists**
6. **Updating font align**
7. **Creating different headings**
8. **Colorizing the background of certain text parts**
9. **A rubber for resetting and clearing the style of text parts**
10. **A reset to clear the complete content**

Edit this page

*Last updated by **Felix***

Previous
« yTextarea

Next
yToggle »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yToggle

Primary — Normal
Primary — Hover
Primary — Focus
Primary — Active
Primary — Disabled

Description
Basic Structure
Properties
Usage
Variations
Setting the yToggle Value
Actions based on the State
Special Picker
SourceLink Picker

Description

The yToggle component is a versatile UI element designed for toggling between two states, such as on/off or true/false. It provides an intuitive way for users to make binary choices with a simple click or tap. The yToggle is perfect for:

- adjusting settings and preferences
- any scenario where a binary choice is appropriate.

Basic Structure

The basic structure of the yToggle includes a track and a thumb. The track serves as the background over which the thumb slides, indicating the current state. Optionally, labels can be added to either side of the toggle to represent the states or provide additional information.

Track

The track can be styled individually and will serve as the background of the toggles main element.

Slide

The slide is the toggles indicator for the state.

Label

The label can display a title and will tell the user what the toggle is used for. There can be either a label on the left or on the right or both. These labels cannot be configured individually, but together at the same time.

Properties

Through its various properties the **yToggle** can be configured to suit your needs. The display below provides you with an overview of all the **yToggle** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic | Style | Toggle | Links | Events

- META
- DISPLAY
- SIZE
- PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yToggle in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

Toggle Elevated Toggle Flat Toggle Outlined Toggle Rounded Toggle Plain

Elevated | Flat | Outlined | Rounded | Plain

generic	> size	> minWidth	95px
generic	> size	> height	30px
style	> background	> bgColor__normal	#44ddbb
style	> shadow	> boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	> border	> borderRadius	8px
style	> label	> label__fontColor__normal	#FFFFFF
style	> track	> trackOff__color__normal	#FFFFFF
style	> track	> trackOn__color__normal	#FFFFFF
style	> thumb	> thumbOff__color__normal	#44ddbb
style	> thumb	> thumbOn__color__normal	#44ddbb

Thumb Styling

Toggle Traffic Light Toggle Neutral Toggle Dark Mode

Traffic Light | Neutral | Black and Yellow

generic	> size	> minWidth	70px
style	> track	> trackOn__color	#4CAF50
style	> track	> trackOff__color	#F44336
style	> thumb	> thumbOn__color	#FFFFFF
style	> thumb	> thumbOff__color	#FFFFFF

Label Positioning

Toggle Label Left Toggle Label Right Toggle Label Both

Left | Right | Both

generic	> size	> minWidth	70px
style	> label	> leftLabel	'left'

Setting the yToggle Value

The yToggle component's value can be dynamically assigned either programmatically via JavaScript or by establishing a link to another component's value—known as a SourceLink. This flexibility allows the yToggle to respond to user interactions or application states in real time.

Via JavaScript

To set the yToggle's value using JavaScript, access the value property within the **"style" > "thumb"** category. This method is ideal for changing the yToggle's state based on user interactions or other events within your application.

Here's how you can set the value in JavaScript:

Toggle EvtClicked

```
// function onToggle_1_EvtClicked (apiObject, component, eventData) {
const desiredValue = true; // true or false
component.set("style", "thumb", "value", desiredValue);
// }
```

Simply replace desiredValue with the appropriate boolean or comparable value that corresponds to the yToggle's position.

Via SourceLink Picker

To link the yToggle's value with another component, use the SourceLink Picker. This feature binds the yToggle's state to another component's value, allowing for integrated component behavior.

Navigate to the SourceLink Picker section in the property settings and select the component you wish to link with. Once configured, the style > thumb > value of the yToggle will reflect the value provided by the linked component, ensuring synchronized states across related UI elements.

This dual approach to value assignment provides a robust solution for managing the yToggle's state and integrating it seamlessly within your application's logic flow.

Alternatively use the SourceLink Picker in the Toolbar as a Toolbar Extension.

Actions based on the State

A custom JavaScript function can be attached to the yToggle to perform an action based on its state. Here's an example that logs the state to the console:

Button EvtClicked

```
// function onButton_1_EvtClicked (apiObject, component, eventData) {
const toggleState = component.get("style", "thumb", "value");
console.log(`The toggle is now ${toggleState ? "ON" : "OFF"}.`);
// }
```

Special Picker

SourceLink Picker

There is a special picker for the yToggle in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of 2 or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Edit this page

*Last updated by **Wegener***

Previous
« yTextbox

Next
yToggleButton »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
yPage
yPopup
yContainer
yImage
yLinkButton
yLabel
yTable
yButton
yCheckbox
yExample
yInput
yQRCode
yRadio
ySelectionPicker
yTextarea
yTextbox
yToggle
yToggleButton
Essentials
yeet Platform
Advanced

yToggleButton

Primary Normal | Primary Hover | Primary Focus | Primary Active | Primary Disabled

Description
Basic Structure
Properties
Usage
Variations
Advanced Variations
Setting the Toggle Options List
Setting the Active Value
Special Picker
SourceLink Picker

Description

The yToggleButton is a versatile user interface component designed to allow users to toggle between multiple options dynamically. This component is especially useful in scenarios where users need to switch between more than two states, such as changing themes, toggling settings, or any multi-state control.

Basic Structure

The yToggleButton includes a main body that acts as the clickable area, with visually distinct states indicated by changes in background color, icon color, and typography. Each state can be customized with its own properties to clearly differentiate between the choices available to the user. Each Toggle Option stands as their own unit and will bring a label and or icon with it.

Label

The label contains the text inside the button. It is normally in the center of the unit, but can also be displayed on the right or left site. By changing the inner padding of the button, the label gets moved too. You can find further explanation at the **Label Property**.

Icon

The icon can be used to show an icon inside a unit of the yToggleButton. You can only choose one Icon per unit. You have to choose if you want to show labels or icons for the whole component. You can find further explanation at the **Icon Property**.

Properties

Through its various properties the **yToggleButton** can be configured to suit your needs. The display below provides you with an overview of all the **yToggleButton** properties.

Properties can be changed directly through three methods:

- Inside the Toolbar, which is positioned in your workarea next to your component where you need it. It shows the most important properties, thus providing a fast and efficient way to configure your component in the most basic way.
- It may also occur in the Toolbar Extension, which is a seamless extension accesible as a dropdown item of the Toolbar. It extends the functionality of the Toolbar by providing advanced pickers for most used properties.
- Inside the Detail Panel, which is located in the righthand drawer. Every property of a component can be configured here.

Generic | Style | ToggleButton | Events

- ▶ META
- ▶ DISPLAY
- ▶ SIZE
- ▶ PLACEMENT

Usage

In this section you'll find a collection of application scenarios and examples that illustrate how to leverage the yToggleButton in ways that deviate from its standard behavior, as defined by yBase. This section aims to inspire and guide you through various possibilities, helping you to implement more complex or unique functionalities tailored to your specific needs. General properties that are universally applicable can be found in the yBase usage section.

Variations

ToggleButton Elevated ToggleButton Flat ToggleButton Outlined ToggleButton Rounded ToggleButton Plain

Elevated | Flat | Outlined | Rounded | Plain

generic	size	minWidth	200px
style	background	bgColor__normal	transparent
style	shadow	boxShadow__normal	0 3px 6px 0 rgba(16, 18, 24, 0.2)
style	border	borderRadius	8px
style	contentNormal	contentNormal__bgColor	#44ddbb
style	contentNormal	contentNormal__iconColor	#FFFFFF
style	contentHover	contentHover__bgColor	#1DA284
style	contentHover	contentHover__iconColor	#FFFFFF
style	contentFocus	contentFocus__bgColor	#1DA284
style	contentFocus	contentFocus__iconColor	#44DDBB
style	contentActive	contentActive__bgColor	#1DA284
style	contentActive	contentActive__iconColor	#FFFFFF

Advanced Variations

For advanced use cases, the yToggleButton can be tailored to fit different design aesthetics and functional requirements. Below are some enhanced variations that showcase the component's versatility:

ToggleButton Icon Only ToggleButton Label Only

Icons Only | Label Only

generic	size	minWidth	200px
style	content	showContentLabel	false
style	contentNormal	contentNormal__bgColor	#F1F3F4
style	contentNormal	contentNormal__fontColor	#6200EE
style	contentHover	contentHover__bgColor	#F1F3F4
style	contentHover	contentHover__fontColor	#6200EE
style	contentFocus	contentFocus__bgColor	#F1F3F4
style	contentFocus	contentFocus__fontColor	#6200EE
style	contentActive	contentActive__bgColor	#6C6E74
style	contentActive	contentActive__fontColor	#3700B3

Setting the Toggle Options List

Setting up the toggle options list and defining the active value are crucial for leveraging the yToggleButton to its fullest potential.

How to:

1. Open Detail Panel: Use the Toolbar to access detailed settings.
2. Navigate to **"Component"** Category: Open the category to adjust component-specific settings.
3. Access "Misc" Group: Here you can find the toggle options list.
4. Edit Toggle Options List: Click on the toggle OptionList to open the configuration window where you can add new entries (value, label, tooltip) or modify existing ones.

Setting the Active Value

The active value can be set in the same **"Misc"** group area inside the **"Component"** category, where a simple input field allows you to define which option is currently active.

How to:

1. Open Detail Panel: Use the Toolbar to access detailed settings.
2. Navigate to **"Component"** Category: Open the category to adjust component-specific settings.
3. Access "Misc" Group: Here you can find the active value editable by an input field.

JavaScript Interaction Example

To programmatically change the active option of a yToggleButton, you can use the following JavaScript snippet:

ToggleButton EvtClicked

```
// function onToggleButton_1_EvtClicked (apiObject, component, eventData) {
component.set("misc", "activeToggleOption", newActiveOption);
console.log(`Active toggle option set to: ${newActiveOption}`);
// }
```

This function retrieves the toggle button component via its unique name, then sets the active option using the API's set method, which updates the component's state accordingly.

Special Picker

SourceLink Picker

There is a special picker for the yToggleButton in the Toolbar as a Toolbar Extension which will be used to link the component to a logic of 2 or more other components. This is a kind of SourceLink Picker, similar to other SourceLink Pickers to be found in other components.

Edit this page

Last updated by ***Wegener***

Previous
« yToggle

Next
Essentials »

What is yeet?
Getting Started
UI Components
Essentials
Spacing
Layout System
SourceLinks
yeet Platform
Advanced

Spacing

Underlying Concept: CSS-BoxModel
Analogy / Example
Content
Margin
Padding
Border

Underlying Concept: CSS-BoxModel

In order to fully utilize the margin- / padding- (and border-) properties you need to understand the CSS-BoxModel. In the following text referred to as Spacing. The Spacing describes the structure of a component. It consists of 4 key parts: Content, Padding, Border and Margin.

Analogy / Example

If you think of your component as a car, the margin, padding and border correspond to the following attributes:

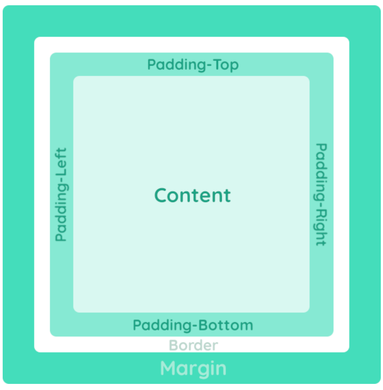
Border: The physical borders of your car, drawing the line between what is the car and what isn't, i.e your chassis / doors or windows.

Margin: The safety distance, the minimum distance which has to be maintained between each vehicle (component) at all times.

Padding: The distance between the driver (content) and your doors or windows (border).

Content

The content of the component includes but is not limited to: text, images, other media and other components.



Margin

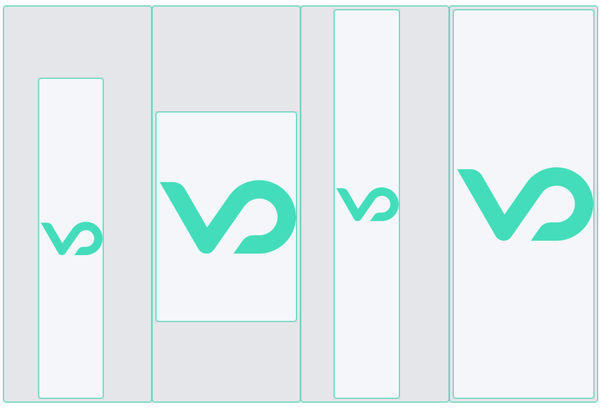
The margin is the minimum distance between the component border and the next component border.

The margin can either be set individually or for each direction (top, bottom, left, right) by adding the corresponding suffix (i.e. margin-bottom instead of margin).

This can be done easil through the use of the **@link** Probertybar1 Spacing Picker:Margin

Common Use-cases include:

- **Positioning** Through specifying a Margin in a certain direction the component is offset my that amount in the opposite direction.
- **Style** Giving each component a small Margin leads to a less crowded Page, which can be perceived as visually appealing.
- **Overlapping** Components can be made to Overlap, this is easily achieved by using negative Margin values.



> TIP
> Margins stack i.e having two components both with a margin of 30 px results in a distance of 60 px between the components.

Padding

The padding defines the distance between component border and component content.

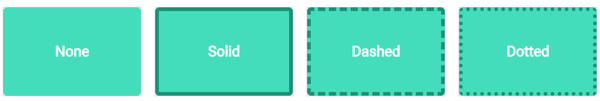
The padding can either be set individually or for each direction (top, bottom, left, right) by adding the corresponding suffix (i.e. padding-bottom instead of padding).

This can be done easil through the use of the **@link** Probertybar1 Spacing Picker:Padding

Common Use-cases include:

- **Style** Preventing Text from reaching the border of a component, increasing readability .
- **Alignment** If you want to Align Images and Texts within a Conatiner.

Border



The border defines the style of the component border. The property consists of 4 subproperties.

Border-style, border-width, border-color and border-radius.

All border properties, except radius, can either be set individually or for each direction (top, bottom, left, right) by adding the corresponding suffix (i.e. border-bottom-style instead of border-style).

This can be done easil through the use of the **@link** Probertybar1 Spacing Picker:Border

Common Use-cases include:

- **Seperator** A component with dimensions like 1px * N px, in conjunction with border-bottom results in a horizontal-styleable seperator, it can also be used vertically.

If you'd like more in depth information regardig the CSS-Boxmodel we can recommend
https://www.w3schools.com/css/css_boxmodel.asp

Edit this page

*Last updated by **Yannic***

Previous
« Essentials

Next
Layout System »

What is yeet?
Getting Started
UI Components
Essentials
Spacing
Layout System
SourceLinks
yeet Platform
Advanced

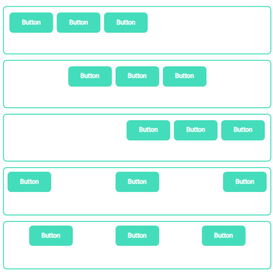
Layout System

Description
Flex sizing in yeet
Main- and Cross-Axis-Alignment
Flex-wrap
Nesting
Layout example

Description

The yeet layout system is based upon the Flexbox-System, but provides a miriyad of simplifications. The main difference can be found in the way as to determine the size of objects. However the following still holds true:

Items are arranged in Rows or Columns.



Items flex (expand) to fill additional space or shrink to fit into smaller spaces.

Every component in yeet is a flex item, additionally the yComponents yPage, yContainer and yPopup always serve as a flex container.

Flex sizing in yeet

The flex property value `(0,1,2,...)` determines the size of the flex item within a flex-container, by putting it in relation to all its siblings. Example: There are 2 flex items within a container, their flex values being `1` and `2` respectively => the spacing ratio is `1:2` => The first item will occupy 33% the second item 66% of the available space. Additionally you can set it `auto`, in this case the item is sized according to its width and height properties, but grows to absorb any extra free space in the flex container, and shrinks to its minimum size to fit the container. Using the value `0` in conjunction with auto results in the component determining its size though its content once during the inital render.

The css properties min-width/-height and max-width/-height can be used in conjunction with the aforementioned flex property. Meaning the dynamic expanding and shrinking of a component is hard constrained within these two bounds.

This combination is the basis for providing a responsive multiplatform web application [SEE LAYOUT EXAMPLE].

Main- and Cross-Axis-Alignment

A main axis has to be defined in the flex-container through the flex direction property defaulting to column, the cross axis is always perpendicular to the main axis, in this case row.

Additionally each axis has an axis start and axis end part. The definition follows the standard western reading flow. Start being left/top while end is right/bottom, depending on the main axis. This can be leveraged in either the justify-content(1) (main axis) or align-items(2) (cross axis) properties.

The following values are most commonly used in either property:

1. **flex-start**
   layout items from start to end
2. **center**
   layout items from the middle on outwards to either side
3. **flex-end**
   layout items from end to start

4. **baseline**
   All flex items are aligned such that their flex container baselines align.
5. **space-between**
   exact same as space-around, except removing the space between outermost item and parent
6. **space-evenly**
   items are distributed so that the spacing between any two items is equal.
7. **space-around**
   evenly distribute all items starting from the center, while giving each item the same amount of space around them, additionaly leaving some extra space between the outermost item and border of the parent container

Flex-wrap

The flex-wrap property specifies whether the flexible items should wrap or not.

- **nowrap**
  Default value. Specifies that the flexible items will not wrap
- **wrap**
  Specifies that the flexible items will wrap if necessary

Nesting

Flex containers can be nested. In this case the child container acts both as a flex item (size, alignment, order rules apply to this item) as well as a container (providing a seperate set of size, alignment and order rules for its children)

Layout example

In order to create the above layout you'd follow these steps:

1. Add two yContainers into an other yContainer and add another container into the lefthand container.
2. Change the flex direction of the lefthand sub-containers into `column` and their justify-content values to `space-evenly`.
3. In the upper-lefthand container: add a yInput for each required input field and change the label of each accordingly.
4. In the lower-lefthand container: add a yCheckbox and a yButton.
5. in the righthand container: add a yInput for each optional input field, as well as a yCheckbox for the newsletter registration.

Edit this page

*Last updated by **Yannic***

Previous
« Spacing

Next
SourceLinks »

What is yeet?
Getting Started
UI Components
Essentials
Spacing
Layout System
SourceLinks
yeet Platform
Advanced

SourceLinks

In order for components to interact within your web app, data exchange is necessary. We provide two methods for achieving this:

UI Snippets:
These allow you to react to events based on JavaScript.

SourceLinks:
For simple use cases, data exchange can occur via SourceLinks. Here, as a user, you only need to specify a data source and its event in the data receiver component. The data will automatically be processed in the receiver component when the specified event is triggered.

Many use cases only require SourceLinks. Let me demonstrate how SourceLinks can be utilized in yeet.

Topbar

Functionality

There are two types of components:

Sender Components: These components send their data to an event bus upon triggering an event.

Receiver Components: They configure themselves to listen for a specific event from a sender component and automatically react to the triggered event of the sender component.

Topbar

Our components can also act as both sender and receiver components. Additionally, a component can respond to multiple events from various senders.

Topbar

Configuring Receiver Components

To enable a receiver component to respond to SourceLinks events, you need to configure the sender component. Begin by selecting a component in the UI designer. Once the component is selected, you can configure SourceLinks via the Toolbar.

You can also find additional SourceLinks options in the Detail Panel under the **"Links"** category. Here, all available SourceLinks for the component are displayed. Clicking on a SourceLink allows you to configure the receiver component using the **SourceLink Picker**.

Sender Components

You can discover which events are triggerable via SourceLinks using the **SourceLink Picker**. When configuring a component, you can determine the events it offers. Additionally, through the **event functions**, you can control whether a SourceLink event should be triggered.

Clipboard

An important feature is the Clipboard, a component designed solely for data transfer. Here, you can fill Clipboard properties with new values via SourceLinks or directly incorporate new values into components via SourceLinks.

Usage

Here are the main use cases for SourceLinks:

Forms

You can easily populate a Clipboard via a Row Select event in the table using a SourceLink. Then, it's straightforward to fill the data of the row in a form on a popup or another page using SourceLinks from the Clipboard. Each component in a form retrieves the initial value from the Clipboard and also writes changes back to the Clipboard via SourceLinks.

Topbar

Text Modules

The label can utilize SourceLink event data as variables through template strings. This allows you to insert simple text modules into your application.

Topbar

Expressions

The filter in the table can also be populated via SourceLinks. Here, template strings are supported, allowing you to easily build dynamic filters.

Topbar

Examples

Some code examples can be found under:

- yLabel:
  Creating a template string using the yInput SourceLink event. Loading table data via a yTable SourceLink event.
- Clipboards:
  Loading data from a table into a clipboard.

Functionality
Configuring Receiver Components
Sender Components
Clipboard
Usage
Forms
Text Modules
Expressions
Examples

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Access Control
Login
Advanced

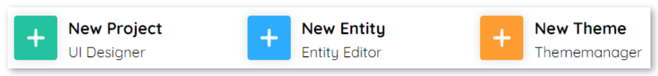
Quickstart
Navigation

Cockpit

The cockpit serves as the entry point into the yeet application, provided the user has successfully authenticated.

Quickstart

The quick links offer the capability to create instances within three main modules of yeet:



- **New project**
  Creates a new frontend project and navigates to frontend > ui.
- **New entity**
  Creates a new entity and navigates to data model.
- **New theme**
  Creates a new theme and navigates to theme editor.

Navigation

On the left side is the navigation of yeet. Here, you'll find all the essential entry points and modules of the application from which you can start your development process.
The navigation is dynamic and adjusts to your current position within yeet. This ensures that the navigation points relevant to you are always available, making navigating within yeet quick and easy.

From the cockpit you can navigate to all necessary ressources that yeet consists of. This includes:

- **Projects**
  To navigate to an overview of all existing frontend projects.
- **Themes**
  To create, read, update and delete all existing themes.
- **Frontend**
  A quicklink to reopen the last opened frontend project.
- **Data**
  To administer entities.
- **Access Control**
  To create, read, update and delete users roles which is only availlable when being on standalone.

Edit this page

*Last updated by **Felix***

Previous
« yeet Platform

Next
Themes »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Theme Manager
Theme Editor
Theme Scraper
Projects
Frontend
Data
Access Control
Login
Advanced

Themes

Description
CoreTheme
SubTheme

Description

In yeet, apps, or rather their components, are visually defined by the underlying theme. This design principle facilitates a consistent visual presentation across various apps by providing a collection of predefined style elements that can be associated with an unlimited number of apps.

CoreTheme

The foundation is formed by what are termed CoreThemes, which, in turn, are composed of various SubThemes. A CoreTheme can be understood as a sort of collection and does not set any style properties by itself. Through the Theme Manager, a new CoreTheme can be easily created with just a click. If necessary, an existing CoreTheme can serve as the basis for this creation.

SubTheme

In contrast, SubThemes provide specific style definitions for different aspects of the user interface. Each SubTheme can be individually tailored to meet the needs and brand identity of the respective app, with the CoreThemes serving as a unified framework within which these customizations take place.
For instance, two different SubThemes can be created within a CoreTheme to style different areas of an application. Buttons in a form, for example, may be intended to look different from how they appear in the rest of the application. In such cases, a new SubTheme can be created and assigned for each variation of the style.

Structurally, the SubTheme can be divided into different areas. Fundamentally, it is composed of:

1. Core Paletts
2. Core Tokens
3. Component Tokens

Edit this page

*Last updated by **Wegener***

Previous
« Cockpit

Next
Theme Manager »

Copyright © 2024 YeetDocs, vectorsoft AG.

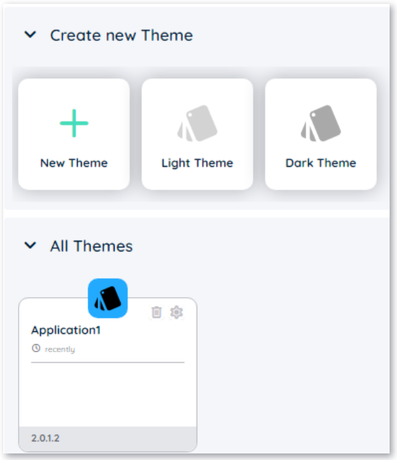
What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Theme Manager
Theme Editor
Theme Scraper
Projects
Frontend
Data
Access Control
Login
Advanced

CRUD CoreTheme

Theme Manager

CRUD CoreTheme

As the name implies, this tool is an overview of all existing themes and offers many possibilities to interact with them.



An unlimited number of CoreThemes can be created. To do this, simply assign a unique name, which serves as the identifier for the CoreTheme. The creation can be done on basis of an existing theme, while using default values when no basis is selected. A theme can be opened by any yeetDeveloper, as long as the version is compatible and springs from the same main version. An existing CoreTheme can easily be renamed in the Theme Manager, but it must not be given a name that is already in use. Finally, it's also the case that any yeetDeveloper can delete CoreThemes, meaning no special access rights are required, nor is it protected by the creator.

Edit this page

*Last updated by **Felix***

Previous
« Themes

Next
Theme Editor »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Theme Manager
Theme Editor
Theme Scraper
Projects
Frontend
Data
Access Control
Login
Advanced

Theme Editor

CRUD CoreTheme
CRUD SubTheme
ContentHeader
Paletts
CoreTokens
Component Overview
ComponentTokens

CRUD CoreTheme

An existing CoreTheme can easily be renamed in the ThemeEditor via the ContentHeader.

If there's a need to save a current theme as a checkpoint, for instance, to create a backup copy, a copy of the current CoreTheme can be saved under a new name using the *"save as"* option.

CRUD SubTheme

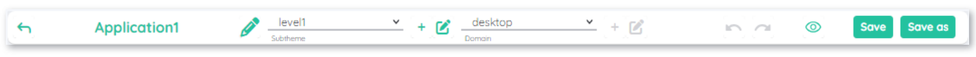
Also through the ContentHeader, a new SubTheme can be created. This can be based on an already existing SubTheme, but if no base is selected, it will always be initialized with default values. The only requirement is a unique name, as it serves as the identifier.

In addition to editing its internal values, the name itself can be changed and determined. The same premise applies here as with the creation of a SubTheme, which is the uniqueness of the name.

To remove erroneous SubThemes or delete obsolete data, a delete functionality is also provided.

ContentHeader

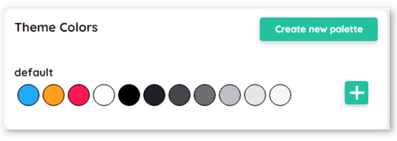
A central component of the ThemeEditor is the ContentHeader. It houses numerous essential functionalities and is the core of the editor.



Thus, it enables the control of CRUD operations on Core- and SubThemes, as well as the ability to undo or redo actions.

Paletts

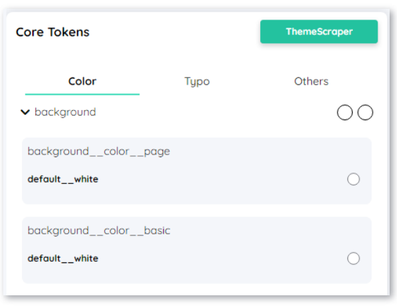
The CorePalettes are color palettes that are part of a SubTheme. An unlimited number of palettes can be created, each of which, in turn, can consist of an unlimited number of color tokens.



The palettes offer the possibility to store recurring or essential colors and create references to them. These references can then be used in numerous places and influence the design of the application.
The significant advantage of these references is that a change in the value of the colors in a palette automatically updates the value of the countless references, significantly reducing the workload.

CoreTokens

The CoreTokens determine the fundamental appearance of my application. They influence the style of the control elements and are thus an essential component of a theme.



When the CoreTokens are altered, my control elements also change, provided that no specific value has been explicitly set via the ComponentTokens or the specific config.
For better clarity, the core tokens are grouped by types and categorized into the three categories **Color**, **Typo** and **Others**.

Inside these Categories, one can find properties like:

- String-Property
- Number-Property
- Color-Property
- UnitNumber-Property
- OptionList-Property

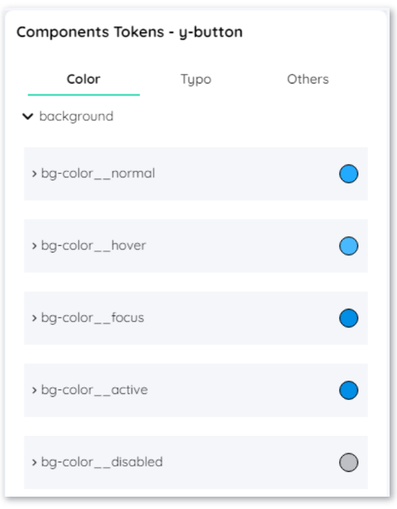
Component Overview

To immediately visualize stylistic changes, the Theme Editor includes an overview of all components and their appearance. The view can be directly filtered, allowing for individual adjustments.

- **Filter by components**
  The view can be pre-filtered to specific component types, creating a better overview.
- **Filter by state**
  In addition to the type of component, the particular state is also filterable. For instance, if only the hover state of components is to be altered, the view can be filtered accordingly.
- **Filter by search text**
  In addition to filtering by component type, this can also be done through search, where both filtering methods are not mutually exclusive.

ComponentTokens

An essential part of theming in yeet are the ComponentTokens. The individual components of the application are influenced in their visual appearance by these tokens. For example, ComponentTokens are used to determine the background color of a yButton.



When the value of a token is changed, it affects all components of the same type, provided the value has not been explicitly overridden in the configuration. When changing the appearance of a certain component type, you can select between choosing a CoreToken, a CustomToken and an individual value.
Similar to the CoreTokens, the ComponentTokens are also categorized into **Color**, **Typo** and **Other** for the same reason.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Theme Manager
Theme Editor
Theme Scraper
Projects
Frontend
Data
Access Control
Login
Advanced

Theme Scraper

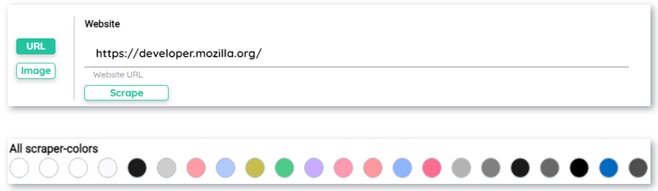
Description
Functionality

Description

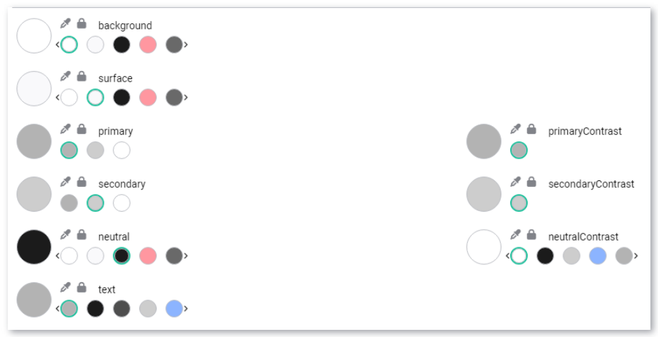
The so-called Theme Scraper serves as a utility tool to quickly and easily adopt the *corporate design* of one's company website, extracting the relevant colors and images.

Functionality

To do this, the tool simply requires a valid **URL** or an **image** to be provided, and the scraper analyzes what is presented to it.



The outcome is the creation of a palette and the distribution of color references across the CoreTokens.



However, before the results are adopted, influence can be exerted on the outcome, and possible adjustments can be made.

Only when satisfaction with the distribution is achieved can the result be adopted and incorporated into the current SubTheme.

It's important to understand the system's two-tier nature. The process begins with the adoption of a palette creation. Once this is done, certain CoreTokens can additionally be overwritten.

This affects:

- **background**
- **surface**
- **primary**
- **secondary**
- **neutral**
- **text**

> INFO
> This also implies that tokens for the statuses **success**, **error**, **info**, and **warning** are not overwritten.

Edit this page

*Last updated by **Felix***

Previous
« Theme Editor

Next
Projects »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Access Control
Login
Advanced

Projects

Project
CRUD
Versioning
Locking
Recent projects
All projects

Project

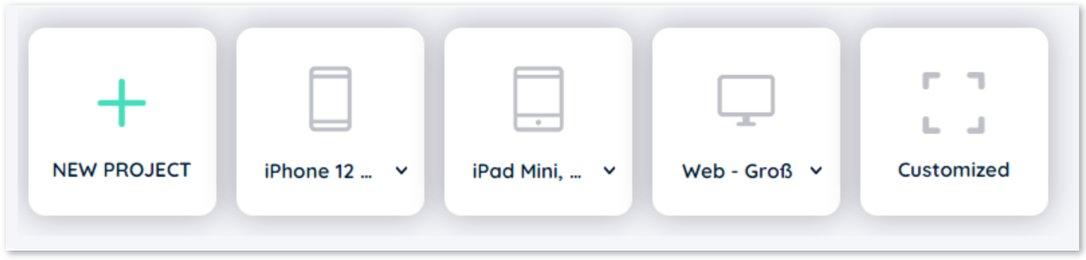
One of the three main resources in yeet, projects represent the instance of your application. A project describes the actual application, namely the interface along with its logic.

As a yeetDeveloper, you can create and manage an unlimited number of projects.

Currently, projects are developed in single user mode, which means that only **one user** can work on a project at any given time. If another user is currently working on your project, they must release it before you can continue working on it. While a project is being edited, it can only be opened and viewed in readonly mode.

CRUD

To create a new project, you must be a yeetDeveloper!



There are five different ways to initiate one:

- **New project**
  Creates a new project in custom format.
- **Mobile**
  Creates a project in smartphone format.
- **Tablet**
  Creates a project in tablet format.
- **Desktop**
  Creates a project in desktop format.
- **Custom**
  Allows for specifying your own breakpoints, on the basis of which a project is created.

Through the dropdowns, a specific preset can be chosen, based on real devices.
Editing the name and project settings is not user specific, meaning, any user can change these properties as long as the project is not locked or set to readonly. Every project that has been created can also be deleted if it is currently available for access.

Versioning

Each project is created based on a certain version. This means that a new release can bring new changes, and your project must first be patched to the new version for successive work! We take care of this, and you only need to open and resave the project once.

Locking

Opening a project or creating a new one, simply results in locking this project. Based on the single user operation, your project cannot be changed by other other users once it is locked. Only the user which claimed the project can release it, by closing it or by actively unlocking it.

This means, a project can be **locked** or **unlocked**.
This is usually indicated by large icons which overlap a project in the overview or indicated by a notification which appears when opening the project in readonly mode.

@Note insert image here

If the project is being overlapped by a lock icon, this usually means that the project is currently locked by another user. The active user is stated beneath the icon with the username. This state means it can only be opened readonly.
Alternatively to the lock icon, a key icon may appear. This usually indicates that you can unlock the project and is only possible, when the current user equals the user that has locked the project.

Recent projects

As the name implies, this page provides an overview of the most recently opened projects and presents them in a summary.

All projects

Here too, the name reveals the scope of the component. It displays all projects available in the database and presents them as a list.

Edit this page

*Last updated by **Felix***

Previous
« Theme Scraper

Next
Frontend »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
Toolbar
Explorer
Detail Panel
UI Logic
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

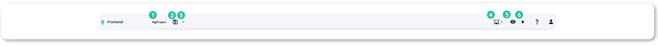
Topbar
Workarea

UI

Here you will find all the tools for creating, testing, and populating your web interface, as well as adding logic.

Topbar

The topmost element is the **topbar**. On the left side, you will find options to change the project name(1) (double-clicking on the project name opens an in-place edit), **save** the current state of your project (2), or **save** the project **under a different name** (3).



On the right side, you can adjust breakpoints to test various device resolutions (4), start the preview (5), or launch the project as a web app (6).

The preview generates a new tab displaying the currently saved project in preview mode. The active page is loaded in preview mode.

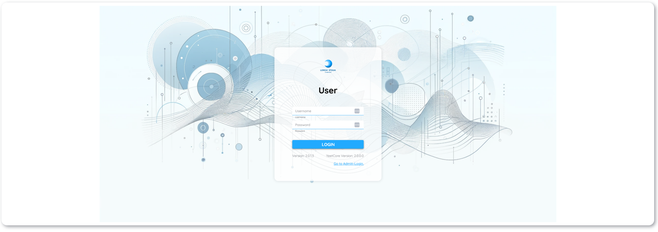
Many functionalities that have been disabled in the UI designer are functional in preview mode. The preview mode serves as a simple testing environment to test the layout and functionality of a page. The Toolbar at the top of the preview also helps in this regard. Here, you can change the project theme (1), use a different breakpoint (2) or load a different page (3) in preview mode.

Preview
Topbar

Every saved project can be launched live directly through a project URL. To do this, you simply need to utilize the project URL, which always consists of:

https://**<domain>**/#/**<de-DE/en-US>**/**<database>**/**<project-name>**/login

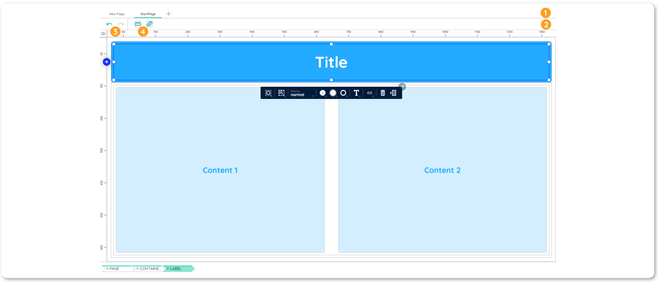
When entering the project URL, we arrive at a login page where we can test the web app live using a test user.



Lastly, on every topbar, you will find the context help (7) and the account settings (8).

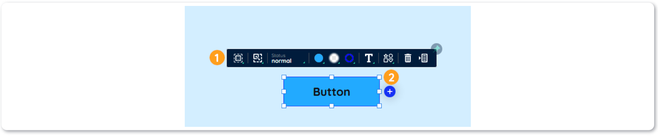
Workarea

Here, we see the active page or popup of our project. Additional pages can be activated through the explorer or selected via the tabbar (1). Below the tabbar, you will find the contentbar (2). Here, you can find the undo and redo buttons (3) for the active page and options for the ruler (4). With the "Hide Rulers" option, you can turn off the rulers, and with the "Hide Frames" option, the component frames will only be displayed for the active or hovered component. This allows you to get a better idea of how your interface will look later on.



The third option allows you to create and display guides. The guides are intended to provide orientation aids for layout design.

Once a page has been activated, we can see all the elements of the page, and the active component is highlighted with a frame containing sizing handlers. Additionally, a context-based Toolbar (1) is displayed, offering various settings for the active component. A copy button (2) is also displayed, allowing you to copy the active component and introduce it into the same container.



Below the displayed page, you will find the Layer View. It always shows the structural path from the main container (yPage/yPopup) to the active component. This allows you to activate components that are not reachable by clicking or better understand the layout structure.

Layer
view

Edit this page

*Last updated by **Felix***

Previous
« Frontend

Next
Toolbar »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
Toolbar
Explorer
Detail Panel
UI Logic
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

Toolbar

One of the most important tools for creating new interfaces is the Toolbar. It is a dynamic bar where you can always adjust the most **important** or **commonly** used properties of a component, while all other component properties are provided in the Detail Panel.

The properties of the component are usually directly adjustable or accessible via a submenu (Toolbar Extension).

To display the settings contextually, there can be a separate Toolbar for sub-elements of a complex component. To do this, double-click on a sub-element to select it, and you can switch back to the main Toolbar via the Back Button.

Toolbar
Subbar

The Toolbar is always displayed for the active component. If it obscures important elements, it can be moved via drag & drop or fixed at a position using the pin button.

Additionally, you'll find a Delete Button (1) to delete the component or the Detail Panel button(2) to toggle the visibility of the Detail Panel.

Toolbar
Subbar

Basepicker

The following pickers will be available for each component:

Status Picker

The State Picker determines the **currently displayed state** of a component. Each component has at least the **normal** state but can also assume other states such as **hover**, **active**, **focus**, **disabled**, **error**, **readonly**, or **disabled**. When a state is activated, only properties relevant to that state are displayed.

Sizing Picker

Through the Sizing Picker, you can determine the dimensions of a component. In the submenu (Toolbar Extension), you will find all properties related to size. You can also adjust the maximum dimensions of a component by dragging. To create good responsive behavior, the interaction between minimum and maximum size values is important. (Flexbox link)

Spacing Picker

To define the spacing around the content (padding) of a component or between components (margin), the Spacing Picker offers adjustment options. Here, you can set the spacing from one or multiple sides simultaneously (1). To roughly define the spacing, you can also adjust distances via drag and drop.

Color Picker

Each component also has a Color Picker for the background color. Each state has its own background color. When clicking on the Color Picker, **Toolbar Extension** opens, where you can adjust the opacity of the selected color (1) or choose a color from the theme color palette (2) and the base color palette (3). With the theme colors, you can switch between existing color palettes in the active SubTheme (4). The limited selection of colors is intended to help you maintain a consistent color style.

Typo Picker

Some components offer labels and texts (e.g., yLabel, yTextBox, yButton, yInput, etc.). All relevant properties for these can be found in the Typo Picker's **Toolbar Extension**. Here, you can adjust the font family (1), font size (2), font style (3), or text color (4). When selecting colors, you are once again limited to the base and theme colors to enforce a consistent style throughout the application.

Border Picker

Another general property group is the border properties of components. Here, each state of a component can have its own defined border. When clicking on the Border Picker (1), you can adjust the thickness (2) and color (3) of the border.

Shadow Picker

To adjust the shadow, the Shadow Bar offers options to define the shadow cast by specifying the **X** and **Y** position (1), adjust the shadow size and density using **Blur** (2) and **Spread** (3), and select a shadow color from the color palettes. The shadow can be defined separately for each state. import StatusPicker from "src/app/common/components/propertybars/propertybar1Picker/StatusPicker.vue"

Edit this page

Last updated by ***Felix***

Previous
« UI

Next
Explorer »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
Toolbar
Explorer
Detail Panel
UI Logic
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

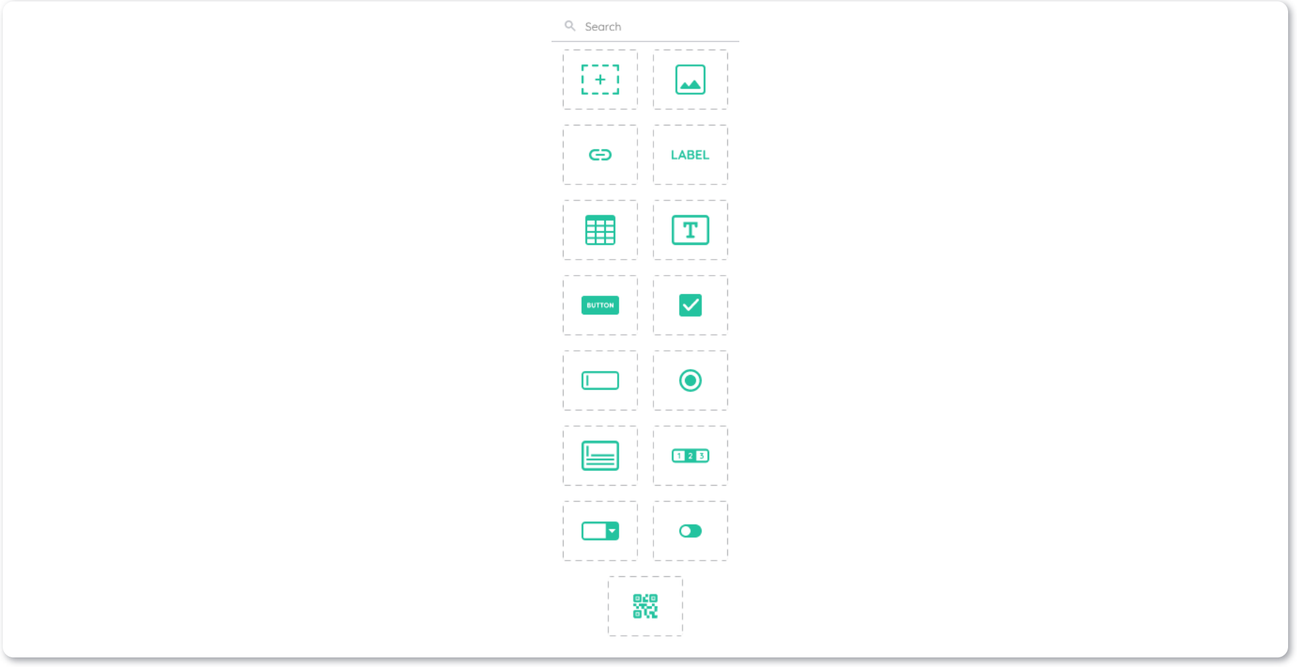
Components
Layout
Overview

Explorer

The Explorer has three sections that can be accessed through tabs or the More Button.

Components

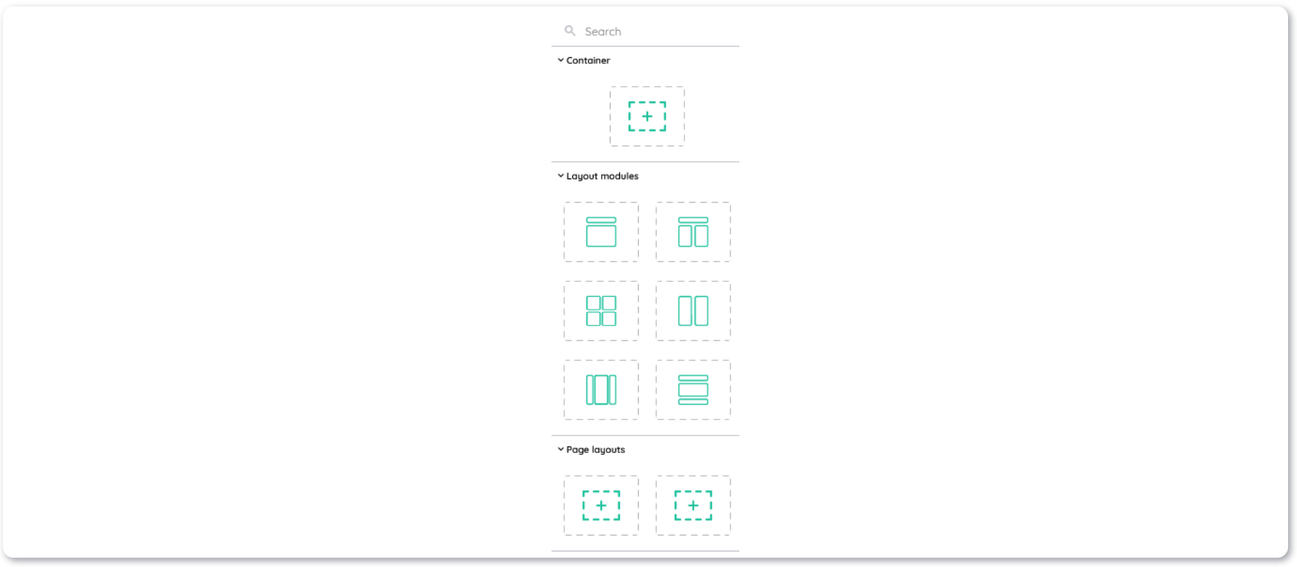
In **Components**, we find all the components available for creating the UI. To do this, we select a **container component** and can then add the component by clicking on the component icon. Alternatively, components can also be added to a page via drag & drop using the component icon.



Layout

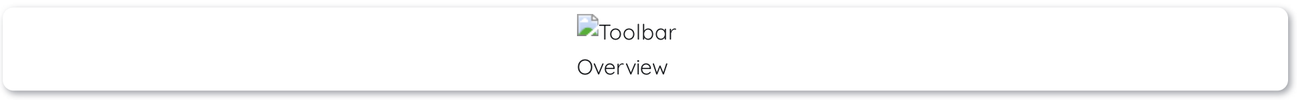
Under Layouts, you'll find various pre-made layout templates. The group of layout modules offers simple foundational templates. These can be combined to create larger, more complex layouts.

Page layouts are templates that include entire forms or pages.



Overview

Here, all popups and pages of the project are listed. The active page or popup is loaded in the work area.



Edit this page

*Last updated by **Felix***

Previous
« Toolbar

Next
Detail Panel »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
Toolbar
Explorer
Detail Panel
UI Logic
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

Detail Panel

Categories
Generic
Style
Component
Links
Events

The Detail Panel displays all adjustable properties of a component. Properties are grouped into context groups and sorted into different **categories**. A detailed description of the properties can be found under Basic Properties and on the component pages. If not all categories can be displayed, all additional categories can be selected via the **"more"** menu.

Categories

Generic

In the **"Generic"** section, you will find all properties that are the same for every component.

- **Meta**
  Here you will find all meta-information about the component.
- **Display**
  Basic rendering settings for the component. (*Currently only flex is properly supported*)
- **Size**
  Properties to control the responsive behavior of the component.
- **Placement**
  Definition of the spacing around other components and additional content.

Style

In the **"Style"** category, all properties are sorted that define the appearance and labeling. Many properties are duplicated because they are set separately for each status. The **status picker** (1) only displays properties that are important for the selected status.

DetailPanelStyle

Another special feature is **themed properties**. All properties that can be defined by a theme can be manually overridden. The override is marked by an **exclamation mark** (1). If you want properties to be determined by a theme again, you can click on the exclamation mark to revert to the theme value.

DetailPanelThemedProperty

Component

Here, properties are sorted that are not directly related to the appearance.

Links

Some components can directly react to specific events of other components. For this purpose, a component is set as the source, called SourceLinks.

Events

Here, you can directly react to component events with JavaScript. When selecting an event, a function can be defined using the UI Snippet Editor.

Edit this page

*Last updated by **Felix***

Previous
« Explorer

Next
UI Logic »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
ApiObject
Events
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

UI Logic

Topbar
Explorer
Snippet Editor
Global Functions
Event Functions

Not all logical functions can be represented through SourceLinks and Clipboards. Therefore, we can define the UI logic using JavaScript snippets. Under UI logic, we find an overview of all the code files in the project. Through the code files, we can edit created event functions or define new global functions.

Topbar

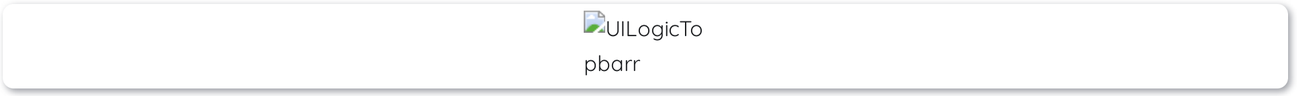
The topmost element is the **topbar**. On the left side, you will find options to change the project name (double-clicking on the project name opens an in-place edit), **save** the current state of your project (2), or **save** the project **under a different name** (3).

On the right side, you can adjust breakpoints to test various device resolutions (4), start the preview (5), or launch the project as a web app (6).

Explorer

In the Explorer, you will find a list of code files. Each code file contains a function that can be utilized by the interface. In the global section, you can define your own global functions. To do this, create a code file folder using the Add button and add your code files to the folder.

On the global level, you may also find other sections. When an event function is created for a component through the Detail Panel, a section is created with the main container name, and a code file is added in a group with the component id. This allows all event functions of a component to be easily found.



Snippet Editor

When a code file is selected, the function header and body are displayed in the editor.

Global Functions

For self-defined functions, the arguments can be freely defined. Global functions are loaded as scripts in the header and are therefore accessible from any event function.

Event Functions

For event functions, the arguments are fixed. The following arguments are passed to each event function:

- **apiObject**
  A collection of useful functions.
- **component**
  Interface object for the current component configuration.
- **eventData**
  Data provided during the event.

Edit this page

*Last updated by **Felix***

Previous
« Detail Panel

Next
ApiObject »

Copyright © 2024 YeetDocs, vectorsoft AG.

Getting Started
Core Concepts
Scripting API
Cloud Services
Editor
Platform
Packages
Manual
Scripting API
ApiObject
ApiCollection
ApiEvent
ApiEnumerations
ApiFunction
ApiInterface
ApiModule
ApiProperty
ApiStruct
ApiTypedef
ApiVariable
ApiObjectFlags
ApiObjectFlagsEnumerations
ApiObjectFlagsFlags
ApiObjectFlagsStatic
ApiObjectFlagsNative
ApiObjectFlagsEditorOnly
ApiObjectFlagsObsolete
ApiObjectFlagsDeprecated
ApiObjectFlagsHidden
ApiObjectFlagsInternal
ApiObjectFlagsPrivate
ApiObjectFlagsProtected
ApiObjectFlagsPublic
ApiObjectFlagsSerializable
ApiObjectFlagsSerializeIfNotDefault
ApiObjectFlagsSerializeIfNotNull
ApiObjectFlagsSerializeIfNotNullOrEmpty
ApiObjectFlagsSerializeIfNotNullOrEmptyOrZero
ApiObjectFlagsSerializeIfNotNullOrZero
ApiObjectFlagsSerializeIfNotZero
ApiObjectFlagsSerializeIfNotZeroOrEmpty
ApiObjectFlagsSerializeIfNotZeroOrEmptyOrNullOrEmptyOrZero
ApiObjectFlagsSerializeIfNotZeroOrNullOrEmpty
ApiObjectFlagsSerializeIfNotZeroOrNullOrEmptyOrZero
ApiObjectFlagsSerializeIfNotZeroOrNullOrZero
ApiObjectFlagsSerializeIfNullOrEmpty
ApiObjectFlagsSerializeIfNullOrEmptyOrZero
ApiObjectFlagsSerializeIfNullOrZero
ApiObjectFlagsSerializeIfZero
ApiObjectFlagsSerializeIfZeroOrEmpty
ApiObjectFlagsSerializeIfZeroOrNullOrEmpty
ApiObjectFlagsSerializeIfZeroOrNullOrEmptyOrZero
ApiObjectFlagsSerializeIfZeroOrNullOrZero
ApiObjectFlagsSerializeIfZeroOrZero
ApiObjectFlagsSerializeNullOrEmpty
ApiObjectFlagsSerializeNullOrEmptyOrZero
ApiObjectFlagsSerializeNullOrZero
ApiObjectFlagsSerializeZero
ApiObjectFlagsSerializeZeroOrEmpty
ApiObjectFlagsSerializeZeroOrNullOrEmpty
ApiObjectFlagsSerializeZeroOrNullOrEmptyOrZero
ApiObjectFlagsSerializeZeroOrNullOrZero
ApiObjectFlagsSerializeZeroOrZero
ApiOperator
ApiOperatorFlags
ApiOperatorFlagsEnumerations
ApiOperatorFlagsFlags
ApiOperatorFlagsStatic
ApiOperatorFlagsNative
ApiOperatorFlagsEditorOnly
ApiOperatorFlagsObsolete
ApiOperatorFlagsDeprecated
ApiOperatorFlagsHidden
ApiOperatorFlagsInternal
ApiOperatorFlagsPrivate
ApiOperatorFlagsProtected
ApiOperatorFlagsPublic
ApiOperatorFlagsSerializable
ApiOperatorFlagsSerializeIfNotDefault
ApiOperatorFlagsSerializeIfNotNull
ApiOperatorFlagsSerializeIfNotNullOrEmpty
ApiOperatorFlagsSerializeIfNotNullOrEmptyOrZero
ApiOperatorFlagsSerializeIfNotNullOrZero
ApiOperatorFlagsSerializeIfNotZero
ApiOperatorFlagsSerializeIfNotZeroOrEmpty
ApiOperatorFlagsSerializeIfNotZeroOrNullOrEmpty
ApiOperatorFlagsSerializeIfNotZeroOrNullOrEmptyOrZero
ApiOperatorFlagsSerializeIfNotZeroOrNullOrZero
ApiOperatorFlagsSerializeIfNotZeroOrZero
ApiOperatorFlagsSerializeNullOrEmpty
ApiOperatorFlagsSerializeNullOrEmptyOrZero
ApiOperatorFlagsSerializeNullOrZero
ApiOperatorFlagsSerializeZero
ApiOperatorFlagsSerializeZeroOrEmpty
ApiOperatorFlagsSerializeZeroOrNullOrEmpty
ApiOperatorFlagsSerializeZeroOrNullOrEmptyOrZero
ApiOperatorFlagsSerializeZeroOrNullOrZero
ApiOperatorFlagsSerializeZeroOrZero
ApiParameter
ApiParameterFlags
ApiParameterFlagsEnumerations
ApiParameterFlagsFlags
ApiParameterFlagsStatic
ApiParameterFlagsNative
ApiParameterFlagsEditorOnly
ApiParameterFlagsObsolete
ApiParameterFlagsDeprecated
ApiParameterFlagsHidden
ApiParameterFlagsInternal
ApiParameterFlagsPrivate
ApiParameterFlagsProtected
ApiParameterFlagsPublic
ApiParameterFlagsSerializable
ApiParameterFlagsSerializeIfNotDefault
ApiParameterFlagsSerializeIfNotNull
ApiParameterFlagsSerializeIfNotNullOrEmpty
ApiParameterFlagsSerializeIfNotNullOrEmptyOrZero
ApiParameterFlagsSerializeIfNotNullOrZero
ApiParameterFlagsSerializeIfNotZero
ApiParameterFlagsSerializeIfNotZeroOrEmpty
ApiParameterFlagsSerializeIfNotZeroOrNullOrEmpty
ApiParameterFlagsSerializeIfNotZeroOrNullOrEmptyOrZero
ApiParameterFlagsSerializeIfNotZeroOrNullOrZero
ApiParameterFlagsSerializeIfNotZeroOrZero
ApiParameterFlagsSerializeNullOrEmpty
ApiParameterFlagsSerializeNullOrEmptyOrZero
ApiParameterFlagsSerializeNullOrZero
ApiParameterFlagsSerializeZero
ApiParameterFlagsSerializeZeroOrEmpty
ApiParameterFlagsSerializeZeroOrNullOrEmpty
ApiParameterFlagsSerializeZeroOrNullOrEmptyOrZero
ApiParameterFlagsSerializeZeroOrNullOrZero
ApiParameterFlagsSerializeZeroOrZero
ApiTypeMemberFlags
ApiTypeMemberFlagsEnumerations
ApiTypeMemberFlagsFlags
ApiTypeMemberFlagsStatic
ApiTypeMemberFlagsNative
ApiTypeMemberFlagsEditorOnly
ApiTypeMemberFlagsObsolete
ApiTypeMemberFlagsDeprecated
ApiTypeMemberFlagsHidden
ApiTypeMemberFlagsInternal
ApiTypeMemberFlagsPrivate
ApiTypeMemberFlagsProtected
ApiTypeMemberFlagsPublic
ApiTypeMemberFlagsSerializable
ApiTypeMemberFlagsSerializeIfNotDefault
ApiTypeMemberFlagsSerializeIfNotNull
ApiTypeMemberFlagsSerializeIfNotNullOrEmpty
ApiTypeMemberFlagsSerializeIfNotNullOrEmptyOrZero
ApiTypeMemberFlagsSerializeIfNotNullOrZero
ApiTypeMemberFlagsSerializeIfNotZero
ApiTypeMemberFlagsSerializeIfNotZeroOrEmpty
ApiTypeMemberFlagsSerializeIfNotZeroOrNullOrEmpty
ApiTypeMemberFlagsSerializeIfNotZeroOrNullOrEmptyOrZero
ApiTypeMemberFlagsSerializeIfNotZeroOrNullOrZero
ApiTypeMemberFlagsSerializeIfNotZeroOrZero
ApiTypeMemberFlagsSerializeNullOrEmpty
ApiTypeMemberFlagsSerializeNullOrEmptyOrZero
ApiTypeMemberFlagsSerializeNullOrZero
ApiTypeMemberFlagsSerializeZero
ApiTypeMemberFlagsSerializeZeroOrEmpty
ApiTypeMemberFlagsSerializeZeroOrNullOrEmpty
ApiTypeMemberFlagsSerializeZeroOrNullOrEmptyOrZero
ApiTypeMemberFlagsSerializeZeroOrNullOrZero
ApiTypeMemberFlagsSerializeZeroOrZero
ApiUnknownValue
ApiUnknownValueEnumerations
ApiUnknownValueFlags
ApiUnknownValueFlagsEnumerations
ApiUnknownValueFlagsFlags
ApiUnknownValueFlagsStatic
ApiUnknownValueFlagsNative
ApiUnknownValueFlagsEditorOnly
ApiUnknownValueFlagsObsolete
ApiUnknownValueFlagsDeprecated
ApiUnknownValueFlagsHidden
ApiUnknownValueFlagsInternal
ApiUnknownValueFlagsPrivate
ApiUnknownValueFlagsProtected
ApiUnknownValueFlagsPublic
ApiUnknownValueFlagsSerializable
ApiUnknownValueFlagsSerializeIfNotDefault
ApiUnknownValueFlagsSerializeIfNotNull
ApiUnknownValueFlagsSerializeIfNotNullOrEmpty
ApiUnknownValueFlagsSerializeIfNotNullOrEmptyOrZero
ApiUnknownValueFlagsSerializeIfNotNullOrZero
ApiUnknownValueFlagsSerializeIfNotZero
ApiUnknownValueFlagsSerializeIfNotZeroOrEmpty
ApiUnknownValueFlagsSerializeIfNotZeroOrNullOrEmpty
ApiUnknownValueFlagsSerializeIfNotZeroOrNullOrEmptyOrZero
ApiUnknownValueFlagsSerializeIfNotZeroOrNullOrZero
ApiUnknownValueFlagsSerializeIfNotZeroOrZero
ApiUnknownValueFlagsSerializeNullOrEmpty
ApiUnknownValueFlagsSerializeNullOrEmptyOrZero
ApiUnknownValueFlagsSerializeNullOrZero
ApiUnknownValueFlagsSerializeZero
ApiUnknownValueFlagsSerializeZeroOrEmpty
ApiUnknownValueFlagsSerializeZeroOrNullOrEmpty
ApiUnknownValueFlagsSerializeZeroOrNullOrEmptyOrZero
ApiUnknownValueFlagsSerializeZeroOrNullOrZero
ApiUnknownValueFlagsSerializeZeroOrZero
OptionsUnknownValue
OptionsUnknownValueEnumerations
OptionsUnknownValueFlags
OptionsUnknownValueFlagsEnumerations
OptionsUnknownValueFlagsFlags
OptionsUnknownValueFlagsStatic
OptionsUnknownValueFlagsNative
OptionsUnknownValueFlagsEditorOnly
OptionsUnknownValueFlagsObsolete
OptionsUnknownValueFlagsDeprecated
OptionsUnknownValueFlagsHidden
OptionsUnknownValueFlagsInternal
OptionsUnknownValueFlagsPrivate
OptionsUnknownValueFlagsProtected
OptionsUnknownValueFlagsPublic
OptionsUnknownValueFlagsSerializable
OptionsUnknownValueFlagsSerializeIfNotDefault
OptionsUnknownValueFlagsSerializeIfNotNull
OptionsUnknownValueFlagsSerializeIfNotNullOrEmpty
OptionsUnknownValueFlagsSerializeIfNotNullOrEmptyOrZero
OptionsUnknownValueFlagsSerializeIfNotNullOrZero
OptionsUnknownValueFlagsSerializeIfNotZero
OptionsUnknownValueFlagsSerializeIfNotZeroOrEmpty
OptionsUnknownValueFlagsSerializeIfNotZeroOrNullOrEmpty
OptionsUnknownValueFlagsSerializeIfNotZeroOrNullOrEmptyOrZero
OptionsUnknownValueFlagsSerializeIfNotZeroOrNullOrZero
OptionsUnknownValueFlagsSerializeIfNotZeroOrZero
OptionsUnknownValueFlagsSerializeNullOrEmpty
OptionsUnknownValueFlagsSerializeNullOrEmptyOrZero
OptionsUnknownValueFlagsSerializeNullOrZero
OptionsUnknownValueFlagsSerializeZero
OptionsUnknownValueFlagsSerializeZeroOrEmpty
OptionsUnknownValueFlagsSerializeZeroOrNullOrEmpty
OptionsUnknownValueFlagsSerializeZeroOrNullOrEmptyOrZero
OptionsUnknownValueFlagsSerializeZeroOrNullOrZero
OptionsUnknownValueFlagsSerializeZeroOrZero
ObjectTypes
gOnDoneEnum
PropertyValueTypes
FrontEndpointObjectTypes
FrontEndpointObjectType
getRuntimeObject
getEnum
getProperty
setEnum
setProperty
getRuntimeProperty
getEnum
setEnum
setProperty
resetProperty

ApiObject

Scripting API object that represents a single object in the scripting API. Every type of ApiObject derivative exposes properties, methods and events.

Obsolete

UI

Scripting API object that offers the following functions:

getObject()

```
ApiObject UI.getObject(int pointer, int[] pointerOffsets)
```

Given a valid ScriptableObject pointer (see the output of GetNativeObjectPtr() for a convenient way to get this pointer) and a set of pointer offsets, this function will return the ApiObject that represents the object at the end of the pointer chain.

Name	Type	Description
pointer	int	The base pointer
pointerOffsets	int[]	An array of pointer offsets, in order. Can be empty.

Returns an ApiObject, or null if:

- the base pointer is invalid
- any of the pointer offsets are out of bounds (e.g. the array is empty)
- the object at the end of the pointer chain is not a ScriptableObject

getNativeObjectPtr(object)

```
int UI.getNativeObjectPtr(object[] pointerOffsetObjectPtr)
```

Given a valid ScriptableObject, this function will return the pointer to the object in question, plus the pointer offsets to the ScriptableObject in question, if any.

Name	Type	Description
pointerOffsetObjectPtr	object[]	The ScriptableObject in question, plus any pointer offsets to the ScriptableObject in question, if any.

Returns an array of two pointers, in order:

- the base pointer
- the pointer offset

actions

create()

```
ApiCollection UI.actions.create(int, int, int,
```

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
ApiObject
Events
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

Events

Many components offer events to which you can react with JavaScript code. To do this, select a component in the work area and open Detail Panel. In Detail Panel, switch to the category Events. Here, you will see all the events that your selected component offers.

Topbar

If you want to attach a function to the event, click on the button next to the event property, and a popup window will open.

Event-Snippet-Editor

In the Snippet Editor, you will find a Explorer on the left side. Here, all event functions assigned to this component are listed.

Topbar

New event functions are only listed in the Explorer after saving. On the right side, you can see the code area and a **content header bar**. In the header bar, you will find controls for undo and redo, as well as the option to call the event functions asynchronously.

Topbar

Below the **content header bar**, you will see the actual Snippet Editor where you can define the function body.

Topbar

Lastly, you can save your changes to the code file (1) or discard all changes (2). To apply your code changes, the project with the modified code files must be saved.

Event-Live-Cycle

- EventTrigger
- EventFunction
- Validate
- Update component config
- Emit SourceLinks

When an event is triggered, a JavaScript function can be defined to be called by the event. If the called function returns false, the default behavior is stopped. This allows, for example, validation to be executed before the new value is applied to the component configuration. Default validations and SourceLink events are also stopped.

Topbar

Arguments

Each event provides the event function with 3 arguments:

- **apiObject**
  A collection of useful functions.
- **component**
  The runtime object for the current component configuration.
- **eventData**
  Data provided during the event.

Runtime objects

To retrieve the current values of the component, we provide a yeetRuntimeObject for simplified access. A RuntimeObject can be created using apiObject.ui.getObject().

```
apiObject.ui.getObject(id: number, type: string, sourceConfigType?: string, sourceConfigId?: number)
=> yeetRuntimeObject
```

- id
  Unique component id.
- type
  Component type.
- (optional)sourceConfigType
  If referencing a component that does not exist on the current active page, specify the component type of the main container (y-page/y-popup) here.
- (optional)sourceConfigType
  If referencing a component that does not exist on the current active page, specify the id of the main container (y-page/y-popup) here.

```
let myInput = apiObject.ui.getObject(1,"y-input", "y-page", 1);
```

Using the RuntimeObject, you can now read (get) or write (set) the desired property via the property path.

The property path always consists of 3 components:

- Category
- Group
- PropertyName

```
let propertyValue = myInput.get("style", "content", "valueString");
myInput.set("style", "content", "valueString", "NewString");
```

The path always follows the property structure in the Detail Panel.

Another way to generate the property path is to use the editor's autocomplete function. Simply enter the component name in PascalCase and navigate through the property structure. Once you've selected your property, the path will be inserted into the editor.

Runtime-Object-Property

If certain properties need to be read and set frequently, you can create a runtime object directly for a property. First, create a runtime object for your component.

```
let propertyValue = myInput.get("style", "content", "valueString");
```

Then, using the runtime object, create a RuntimeObjectProperty as follows:

```
property(category: string, group: string, name: string): yeetRuntimeObjectProperty
let inputValueStringProperty = myInput.property("style", "content", "valueString");
```

Now, you can easily access the property using **get** and **set** methods.

```
let value = inputValueStringProperty.get();

inputValueStringProperty.set("NewValue");
```

Event-Snippet-Editor
Event-Live-Cycle
Arguments
Runtime objects
Runtime-Object-Property

Edit this page

*Last updated by **Felix***

Previous
« ApiObject

Next
Clipboards »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

Topbar
Sidebar
Content
Clipboard-Properties
Entity-Properties

Clipboards

Clipboards offer a method for maintaining data across various pages by defining a Clipboard along with its freely chosen properties. These properties can be populated with data through SourceLinks or the UI snippet function. Furthermore, the data within these properties can be directly integrated into components using SourceLinks or retrieved using the UI snippet function. This facilitates the easy transfer or modification of form data to another page. Moreover, a Clipboard can be directly filled with data from an entity, provided that the Clipboard's structure matches that of the chosen entity. Through specific SourceLink events, an entity record can be created, loaded, altered or deleted using its RecordID.

Clipboards are accessible via the submenu under "Frontend." Each project has its defined Clipboards.

Topbar

Topbar

In the Clipboard editor, at the top, you will find the **topbar**. On the left side, there are options to **change**(1) the project name (double-clicking the project name opens an in-place edit), **save the current state** of your project (2), or **save** the project under a different name (3).

On the right side, you can launch the preview (4) or start the project as a real web app (5).

Topbar

Sidebar

In the **sidebar**, you'll find a list of all previously defined Clipboards. Additional Clipboards can be defined using the Add button (1). The name of a Clipboard can be changed by double-clicking on its label (2). The Clipboard ID is assigned by the system itself and is always unique to a project. If you want to delete a Clipboard, there's also a Delete Button (3).

Topbar

Content

Clipboard-Properties

Clicking on an entry in the list takes you to the different sections of the Clipboard. When you click on the Data-Properties, you will see all the defined properties in the content area.

Topbar

Expanding a property allows you to change the property name, set the value type, and define one or more SourceLinks to load data into the property.

Topbar

If you want to add more properties, you can create additional properties by clicking the Create button.

Topbar

Entity-Properties

When the "Use Entity" checkbox is activated, you can make settings for communication with entities from the database through Entity Properties.

Topbar

Here, you can select the entity with which the Clipboard should communicate (1) and define a record id to be used initially for communication with the entity.

For the Clipboard to interact with the entity, we offer functions through the ApiObject, or you can execute specific actions via SourceLinks. You can set up one or more SourceLinks for the following events (2).

Topbar

Select Row

This allows you to respond to the Select-Row event in tables. When a select event is triggered, the data is directly assigned to Clipboard properties with the same name and then immediately passed on to all components that have specified the Clipboard as their source in a SourceLink through the input event.

> INFO
> Currently, this is only supported by the yTable.

Commit Create

When the Clipboard catches a Commit-Create event through the SourceLink, it attempts to create a new record in the selected entity using the data from the properties.

> INFO
> Currently, this is only supported by the yButton and yLinkButton

Commit Read

When the Clipboard catches a Commit-Read event through the SourceLink, it attempts to load the record with the RecordID (_id) using the Key property and save the data into properties with the same name.

> INFO
> Currently, this is only supported by the yButton and yLinkButton

Commit Update

When the Clipboard catches a Commit-Update event through the SourceLink, it attempts to save the data in the properties into the record with the **RecordID** (_id) using the Key property.

> INFO
> Currently, this is only supported by the yButton and yLinkButton

Commit Delete

When the Clipboard catches a Commit-Delete event through the SourceLink, it attempts to delete the record with the **RecordID** (_id) using the Key property, aiming to remove the data associated with the record.

> INFO
> Currently, this is only supported by the yButton and yLinkButton

Docs
Tutorial

Community
Stack Overflow
Discord
Twitter

More
Blog
GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
Clipboards
Navigations
Navigationtree
Masks
Mask Elements
Projectsettings
Data
Access Control
Login
Advanced

Navigations

In the **"Navigation"** subsection, you can define the navigation for your project. To do this, you create a navigation structure based on a **navigation tree** and display it as a menu in your project using a **navigation component**. We offer simple base navigations that can be customized using additional components.

Navigation can be selected via the submenu under **"Frontend"**. Each project has its own navigation defined.

Topbar

Topbar

In the Clipboard editor, at the top, you will find the **topbar**. On the left side, there are options to **change** (1) the project name (double-clicking the project name opens an in-place edit), **save the current state** of your project (2), or **save** the project under a different name (3).

On the right side, you can launch the preview (4) or start the project as a real web app (5).

Topbar

Explorer

In the explorer, you can switch between the Navigation Tree and Masks modes using the tabs. Depending on the active tab, either the navigation structure editor or the mask editor will be displayed.

Topbar

Content area

Here, a preview corresponding to the current selection in the Explorer is displayed.

Detail Panel

Here, properties can be adjusted for the current Explorer selection.

Topbar

Topbar
Explorer
Content area
Detail Panel

Edit this page

*Last updated by **Felix***

Previous
« Clipboards

Next
Navigationtree »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
Clipboards
Navigations
Navigationtree
Masks
Mask Elements
Projectsettings
Data
Access Control
Login
Advanced

Tree
Detail Panel

Navigationtree

The navigation structure is defined using a tree. Each possible route in the navigation is represented by a node in the tree. To adjust the navigation structure, select the **"Navigation Tree"** tab in the Explorer within the navigation editor.

Tree

When the "Navigation Tree" tab is selected in the Explorer, the tree is displayed in the Explorer. When creating a new page, a node is automatically added to the navigation tree.

You can create additional nodes using the Delete Button (1) and the Add Button (2).

Topbar

You can use drag & drop to arrange the order of navigation entries or to place items into submenus.

When a navigation type is selected, a preview is displayed in the content area.

Detail Panel

When a navigation entry is selected, a Detail Panel opens on the right side. Here, you can set the following properties:

- Label
Display name in the navigation.

- SourcePageId
Specifies which page to load.

- Navigationtype
Determines which mask to use to display the navigation. More information is available under **Masks**.

- Is default page
Specifies which page to show after logging into the app. Only one default page can be set.

- Menu entry hidden
Hides the navigation entry in the navigation.

- Full Path
When specifying the route, the entire path must be provided. Usually, the automatically generated URL mirrors the hierarchical structure of the navigation, with each level in the structure forming part of the route entry chain. For simpler URLs, custom URLs can be defined using the Full Path option.

- Route
The path that appears in the browser URL when the node is clicked. Depending on whether the entire path needs to be specified, we may only need to provide the last part of the URL path. Otherwise, the rest will be automatically generated from the tree structure.

> **INFO**
> If you want to define aliases for specific pages, you can add additional nodes here with URLs that point to the same page.

Topbar

- Group
Some masks offer a grouping function for navigation entries. If certain entries are to be grouped, the Group property can be used to determine which entries belong to a group. The navigation component iterates through the entries in their order and changes the style depending on the Group value.

Topbar

Edit this page

*Last updated by **Felix***

Previous
« Navigations

Next
Masks »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
Clipboards
Navigations
Navigationtree
Masks
Mask Elements
Projectsettings
Data
Access Control
Login
Advanced

Mask list
General
Special

Masks

Above a mask, we define the display of the navigation tree. Therefore, we offer fixed basic navigations. In addition to the fixed basic navigations, we can create our own navigations. To do this, we select **Masks** on the left side in the Explorer.

Mask list

When the **Masks tab** is **activated** in the Explorer, you'll see an overview of all defined masks for your navigation. You can create your own masks using the Add button (1). To do this, create a new mask with the Add button, and a **new entry** will appear under **burger-menu**.

By doing this, based on the burger menu mask, you have created a new mask. When you select the **new mask**, a preview of the mask is displayed again, and on the right side, you will find a sidebar with categories to further determine the appearance of the mask.

General

In the **General** category, you will find properties that are the same for all masks.

- Label
Mask Name.

- Navigationtype
Selected base mask. The property can be changed afterward. All properties in the Special category will be reset when switching the base mask.

- Item-Position
Depending on the base mask, the navigation entries can be displayed in various locations. Possible positions are:

- Topbar
- Left Sidebar
- Right Sidebar

- Bar-Element
Determines the element color of the mask elements. Initially, themed is always selected. Themed means that the **primary_color_main** from the default Project **Theme** is used as the color here. Clicking on the property will offer a color picker to set a fixed color. The reset button can be used to revert to the theme color.

- Bar-Background
Determines the background color of the mask. Initially, themed is always selected. Themed means that the **background_color_basic** from the default Project Theme is used as the color here. Otherwise, behaves the same as the Bar Element property.

- Logout-Button
Adds a logout button to the navigation.

- prefixContainer/ centerContainer/ suffixContainer
Each mask offers 3 slots where you can display additional content. Using a dropdown menu, select an element to be added, and use the Add button to add the element to the slot. Now you will see an entry in the slot list where you can adjust additional properties for the element. Currently, we offer the following elements:

- navigation-label
- navigation-logo
- navigation-link

Special

The Special category depends on the base mask. Here, properties are displayed that are additionally required for the different masks.

Button-Navigation

In the button navigation, only the first level of the navigation structure can be displayed. Entries are displayed as a button group in a topbar.

Special Properties

- **Underline Elements**
  Displays the active navigation point with an underline.

Burger-Menu

In the burger menu, the navigation structure is represented using a popup menu. This allows for easy navigation through deeper navigation structures.

Special Properties

- **Headline**
  In banner mode, the entered value is displayed as a title in the top section of the popup.
- **Headline-Image**
  Here, a image URL can be specified. In banner mode, the image is used as a background image for the title in the popup. If banner mode is not active, the image is centered in the top section of the popup (e.g., for a company logo).

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
Clipboards
Navigations
Navigationtree
Masks
Mask Elements
Projectsettings
Data
Access Control
Login
Advanced

navigation-label
navigation-logo
navigation-link

Mask Elements

Every mask can be customized with additional elements. Here, you will find all the addable mask elements and their properties.

navigation-label

An element to display static text.

- **Name**
  Defines the element name.
- **Height**
  Specifies the height in pixels or percent.
- **Width**
  Specifies the width in pixels or percent.
- **Label**
  Display text.
- **Font-Size**
  Font size of the display text.
- **Font-Color** Font color can be assigned here directly via the color picker or determined through the theme (label__label__font-color__normal).

navigation-logo

An element to visualize images or logos.

- **Name**
  Defines the element name.
- **Height**
  Specifies the height in pixels or percent.
- **Width**
  Specifies the width in pixels or percent.
- **Url**
  URL that points to an image on the web.

navigation-link

An element used to display links.

- **Name**
  Defines the element name.
- **Height**
  Specifies the height in pixels or percent.
- **Width**
  Specifies the width in pixels or percent.
- **Label**
  Name of the link.
- **Url**
  Target URL of the link.
- **Font-Size**
  Font size of the display text.
- **Font-Color**
  Font color can be directly assigned here via the color picker or determined through the theme (label__label__font-color__normal).

Docs
Tutorial

Community
Stack Overflow
Discord
Twitter

More
Blog
GitHub

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
UI
UI Logic
Clipboards
Navigations
Projectsettings
Data
Access Control
Login
Advanced

Projectsettings

The Projectsettings can be accessed by the project manager through the project tiles, or via the frontend by navigating to Projectsettings in the navigation bar.

General properties

In the Projectsettings, we find general properties such as project image, project name, or project description.

Topbar

Themes

Furthermore, we also find the Themes section here. By clicking on the theme label, we can access them.

Topbar

Default Theme

In the Themes section, we have the option to specify the Main Theme and select a SubTheme to load from the second dropdown list. The SubTheme will then be loaded whenever the project is accessed.

Topbar

Project Themes

In project themes, we can assign additional themes to the project, allowing for easy switching between themes at any time within the project.

> **INFO**
> When creating a new theme, it must first be added to the project themes list before it can be used in the web application.

Topbar

General properties
Themes
Default Theme
Project Themes

Edit this page

*Last updated by **Felix***

Previous
« Mask Elements

Next
Data »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Data Model
Data Elements
CSV Import
CSV Export
Access Control
Login
Advanced

Data

Locking Behavior
Entity
Enum

Data is the hub where you manage all aspects of your information. Here, your data is categorized into two main types: entities and Enums. The "Data" section is the foundational element of our platform, designed for defining, managing, and manipulating all data structures and elements. This pivotal section is structured into several key subsections: Entity Editor, Data Elements, CSV Imports, and CSV Exports. Each offers distinct functionalities to streamline your data management tasks.

In the platform's left sidebar, under the "DataModel" category, you're presented with the option to navigate between enums and entities. This allows you to actively select and work with either an entity or an Enum based on your specific needs. Activating an entity or Enum is straightforward and intuitive, providing you with immediate access to either data type for editing or review.

It's important to note that the choice between enums and entities is exclusively available in the "DataModel" category within the sidebar. In all other categories, the sidebar is designed to permit the selection of entities only. This is because, within these contexts, offering Enums would not be applicable or meaningful. The sidebar's design ensures that you are always presented with relevant options that align with the specific functionalities and requirements of each section of the platform.

Locking Behavior

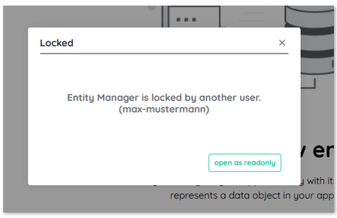
In our platform, the "Data" section is safeguarded by a sophisticated locking mechanism, designed to ensure the integrity and security of your data while it's being accessed or modified. This feature is crucial for maintaining a streamlined workflow and preventing conflicts or unauthorized changes.

The locking mechanism operates on a simple yet effective principle: only one user can make changes within the "Data" section at any given time. This is achieved through a backend lock that activates as soon as a user enters this section. The lock serves as a protective barrier, granting exclusive access to the user and blocking concurrent modifications from others.

Once a user finishes their work and exits the "Data" section or logs out, the lock is automatically released. This ensures that the lock is held only while necessary and is promptly freed up for others to use. However, if a user navigates away from the "Data" section without properly exiting or closes their browser tab unexpectedly, the lock remains active. This persistence is a critical safeguard against potential data breaches or accidental changes, ensuring that the data remains protected until the user formally releases the lock.

In situations where another user encounters locked data, access can be regained in two ways: either by being the original user who applied the lock or by having administrative rights that allow for unlocking. This dual approach ensures that data can always be accessed by those with the right permissions, while maintaining strict control over who can make changes.

For those without the permissions to unlock or who are not the original user, a read-only mode is available. This mode ensures continued access to the data, allowing users to view but not alter the information, thus maintaining data security and integrity.

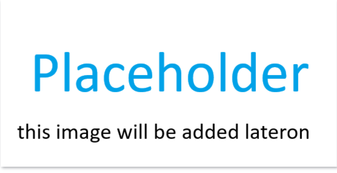


It's also worth noting that the CSV Import functionality incorporates an additional layer of access restrictions, further refining the control over data manipulation. This specialized lock mechanism is detailed in the CSV Import section, highlighting our platform's commitment to data security.

By implementing this locking behavior, our platform not only protects data from unauthorized access or accidental modifications but also ensures that users can work in an environment where data integrity is paramount. This methodical approach to data security underscores our dedication to providing a robust and reliable data management system.

Entity

Entities are the building blocks of your data models. Each entity represents a specific type of data within your application and defines the structure of this data through a set of attributes or fields. Entities enable the modeling of complex data structures and relationships that form the basis of the logic and functionality of your applications. By carefully defining your entities, you lay the foundation for efficient data manipulation, searching, and evaluation in your low-code application.



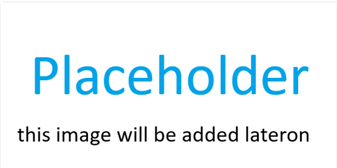
In our platform, entities are structured with various column types, tailored to specific data handling needs. Here's a concise guide to understanding these types and the input mechanisms associated with them:

- **Id**
  Reserved for unique identifiers, ensuring each record in your entity is distinct.
- **String**
  For text-based inputs, accommodating both letters and numbers for versatile data entries.
- **Numerical Types**
  These include Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Decimal, and Double, each designed for numerical data with varying capacities. A specialized number picker restricts inputs to numeric values, facilitating precise data entry for quantitative fields.
- **Boolean**
  Utilizes a straightforward checkbox to capture true or false values, simplifying binary decisions.
- **Date and Time**
  Both employ pickers tailored to their respective data types, allowing for intuitive selection of dates and times. These fields are nullable, featuring an additional checkbox for selecting null. When null is chosen, the picker is hidden, indicating the absence of a value.
- **Link**
  Dedicated to URL or hyperlink entries, this field is also nullable, ensuring flexibility in data representation.
- **Enum**
  Enumerations provide a dropdown selection from a predefined list of options. This type is nullable as well, allowing for a clear distinction when no selection is made.
- **Computed**
  A unique field for values derived from other data points or calculations within your entity. Similar to Enums, Computed fields are nullable, accommodating dynamic data scenarios.

For nullable fields such as Date, Time, Link, Enum, and Computed, the UI adapts to highlight the option for null values. This is manifested by a null checkbox which, when selected, conceals the standard input picker or field, underscoring the choice of not setting a value. This mechanism ensures users can explicitly indicate the absence of data, enhancing clarity and precision in data management.

Enum

An Enum (Enumeration) defines a set of named constants that represent a specific group of values. Using Enums, you can provide clearly defined choices in your entities, enhancing data integrity and user experience through straightforward selection options. Enums are perfect for representing states, categories, or other fixed data sets that do not change.



> **INFO**
> The Enum section is not fully developed yet and just a visual preview. There is no real functionality behind it!

Edit this page

Last updated by ***Miqueas***

Previous
« Projectsettings

Next
Data Model »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Data Model
Data Elements
CSV Import
CSV Export
Access Control
Login
Advanced

Data Model

Entities
Enums
Edit and Commit System
Indicators and Actions

The Entity Model section within our platform offers a robust and intuitive interface for managing your data structures. Within this section, you have the flexibility to navigate between entities and enumerations (enums) depending on your requirements. This choice can be made from the left sidebar under the "DataModel" category, which houses both "Entity" and "Enum" tabs.

Entities

When you opt to work with entities, the platform allows you to either select an existing entity from the sidebar or create a new one by clicking the "plus" icon. Once an entity is selected, you can proceed to customize it according to your project's needs.

Sidebar

NewEntity

In the content area, you're presented with a list of properties that you can add to your entity. You can add a property by either clicking on it from the list or using the drag-and-drop feature, after which you can give it a suitable name. However, any modifications made to an entity, such as adding, renaming, moving, or deleting properties, require you to first "apply" these changes via the topbar. Following this, you must "commit" the changes through the commit bar located in the sidebar to finalize them. The different properties that can be added are explained in the data-section.

It's important to note that editing an entity's properties, including renaming, moving, or deleting, is only possible as long as no data is associated with that entity. This ensures the integrity and consistency of your data.

Enums

Switching to enums offers a more streamlined experience. You can add a new enum by clicking the "plus" button, which prompts you to enter a name, value, and description for the enum. Similar to entities, any changes or additions to enums must be applied through the topbar and then committed using the sidebar commit button.

> **INFO**
> The enum-section is not fully finished yet. This is rather a preview for what is to come soon.

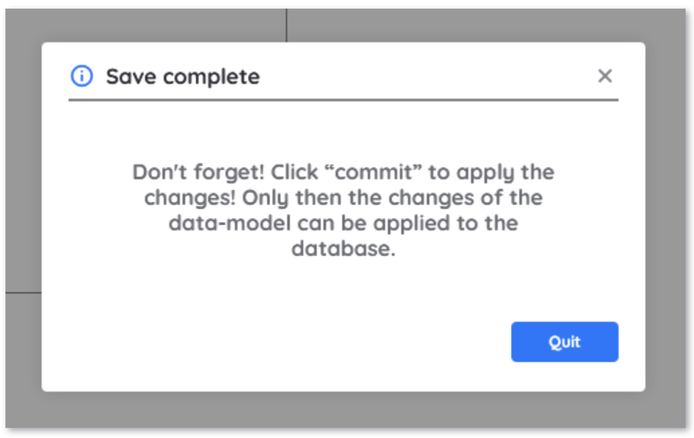
Edit and Commit System

This section explains the universal system for editing and committing changes across both entities and enums, designed to offer flexibility while ensuring data integrity and coherence within your application:

- **Applying Changes**
  In the content area, multiple changes can be made simultaneously. To make these changes official, use the control buttons in the header bar: a trash can icon to undo all recent changes, a redo button to revert the last action, an "apply changes" button to test all modifications on a model, and an arrow button to edit descriptions.

  Header

- **Committing Changes**
  After "applying changes," the platform temporarily implements your modifications for review. Changes need to be "committed" through the commit bar in the sidebar to become permanent.



Commit

Indicators and Actions

- A star next to an item in the sidebar indicates new additions.
- Red signifies errors that prevent committing, indicating non-commitable changes due to errors or invalid references.

Invalid

- Green denotes changes that are valid and ready for committing.
- Orange may indicate a dependency on an entity or enum that doesn't exist, thus invalid, or that an entity is linked but not yet created.
- Grayed-out items indicate deletion in the last applied changes session.

ValidDelete

- Editing restrictions apply when data is associated with an entity, limiting the ability to modify existing properties. However, new properties can always be added to the end, and within the same "apply changes" session, it's possible to add, rename, and adjust properties without committing them immediately.

Whether adding new elements or adjusting existing ones, these tools and guidelines help manage your data structures efficiently and effectively.

Edit this page

*Last updated by **Wegener***

Previous
« Data

Next
Data Elements »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Data Model
Data Elements
CSV Import
CSV Export
Access Control
Login
Advanced

Data Elements

The Data Elements section stands as a pivotal area within our platform, offering you nuanced control over the attributes of your entities. This section is instrumental in defining detailed properties, setting validation rules, and specifying default values for each data element, all of which are fundamental for upholding data integrity, complying with precise data storage needs, and enabling efficient data processing.

Core Functionalities
Data Entry and Modification
Tailored Input Fields by Data Type
Managing Data Modifications
Enhanced Data Management
Dynamic Data Management Controls
HeaderBar and Sidebar Interactions
Comprehensive Filter Functionality

Core Functionalities

Data Entry and Modification

The interface is intuitively designed to facilitate the quick addition and editing of data rows. An "Add" button, prominently located in the header, simplifies the direct insertion of new data entries, complementing the capabilities offered by the CSV Import feature.

Tailored Input Fields by Data Type

For a seamless data entry experience, the system provides specific input fields adapted to various data types:

- **Numerical Types (Byte, Int16, Int32, Int64, UInt16, UInt32, UInt64, Decimal, Double)**
  These come with number pickers for numeric-only inputs.

  Number

- **String, Boolean, Date, Time, and Link**
  Designed for text entries, binary choices, and dates/times/URLs, respectively, with Date, Time, and Link being nullable fields.

  String

  Nullable

  NullableNull

- **Enum and Computed Fields**
  Allow selections from predefined lists and accommodate calculated values. These fields are also nullable to offer flexibility in data representation.

Managing Data Modifications

Modifying entity attributes is streamlined into a clear process. Initial changes are made in the content area, requiring confirmation via the header's "Apply Changes" button. Following this provisional step, you must commit your changes using the sidebar's commit button to solidify your modifications into the entity's schema. Review and Adjust Imported Data

Beyond adding and customizing data, the Data Elements section provides a platform for inspecting and tweaking data previously imported through the CSV Import functionality. This ensures that your application's data remains precisely aligned with your operational needs.

Enhanced Data Management

Dynamic Data Management Controls

The system's dynamic nature allows for the addition, renaming, and reordering of entity properties directly within the platform. Such modifications, however, are tentative until you commit them, emphasizing the critical role of the commit action in finalizing data adjustments. An overview over the types available can be found in the odata-section.

HeaderBar and Sidebar Interactions

The HeaderBar offers a suite of controls for data management, including the ability to insert new data rows, clear existing data, and apply filters for enhanced data navigation.

HeaderAll

Upon editing a data row, the HeaderBar updates to display options for saving or discarding changes, ensuring data integrity.

HeaderRow

Comprehensive Filter Functionality

The platform's filter feature employs oData Filter Operators to facilitate precise data querying. This advanced function, detailed in external documentation, allows you to apply filters based on specific criteria, with the type of filter operator varying according to the data type of the column.

Filter

Edit this page

*Last updated by **Miqueas***

Previous
« Data Model

Next
CSV Import »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Data Model
Data Elements
CSV Import
CSV Export
Access Control
Login
Advanced

CSV Import

Entity Selection and File Upload
Preview and Settings
Import Log and Controls
Column Mapping: Bridging Data with Entities
Lock and Claim System: Managing Concurrent Imports

The CSV Import feature is a pivotal tool within our platform, designed to streamline the process of integrating external data into your application. By selecting the "CSV Import" function, you embark on a straightforward journey to populate your entities with data from CSV files, enhancing your application's data richness and functionality.

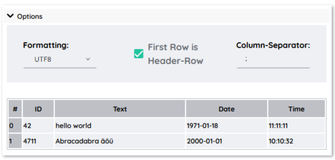
Entity Selection and File Upload

The journey begins in the left sidebar, where you are required to select the target entity for your data import. This step ensures that the data you import aligns with the structure of the chosen entity, facilitating a smooth integration process.

Upon selecting an entity, the central area presents you with a file selection picker, allowing you to upload a CSV file. It's important to note that the system only accepts files with a ".csv" extension, ensuring compatibility and preventing errors during the import process.

Preview and Settings

Following the file upload, a "preview" section appears next to the file picker in the content area. This preview, displaying up to 5 lines from your CSV file, offers a glimpse into how the imported data will appear within the table structure of your application. It's a crucial step for verifying the data's organization and ensuring it meets your expectations before finalizing the import.

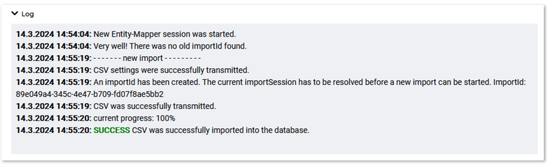


Above the preview, you'll find a settings area where you can customize how the system interprets your CSV file. These settings include:

- **Encoding Format**
  Choose between UTF-8 or ASCII encoding to match the format of your CSV file, ensuring correct interpretation of the data.
- **Header Row Identification**
  Allows you to specify if the first row of your CSV file serves as a header, which will not be imported as data but used to identify column names.
- **Row Separator Character**
  Define the character used to separate rows in your CSV file, such as commas or semicolons. This setting ensures that the preview and subsequent import accurately reflect the structure of your data.

Import Log and Controls

The content area's lower section hosts a log, dynamically updating you with messages about the import process, including success notifications, errors, and warnings. This log also displays the import's progress in percentages and indicates if the import is locked, completed, or requires release.

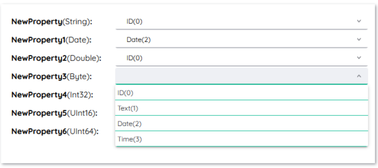


To manage the import process, the content header above the preview area houses crucial controls. These include:

- **Send to Database**
  Initiates the import process when no current import is underway.
- **Rollback**
  Available if an import has either completed or is in progress, allowing you to revert the import actions.
- **Resolve**
  Appears alongside Rollback, enabling you to address any issues encountered during the import and proceed with finalizing the data integration.

Column Mapping: Bridging Data with Entities

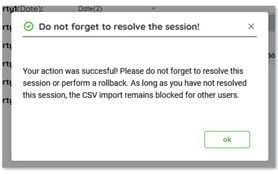
After uploading your CSV file and adjusting settings to suit your data structure, the next critical step involves mapping the CSV columns to the corresponding fields of the selected entity. This process is facilitated by the Column Mapper, located directly below the file upload picker in the content area.



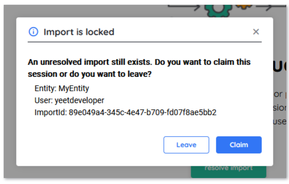
The Column Mapper is your tool for specifying which column from the CSV file corresponds to which field in the entity. This step is vital as it ensures that the imported data aligns correctly with the entity's structure. Notably, the system allows you to map a single CSV column to multiple entity fields if required, providing flexibility in how data is integrated into your application. Before proceeding with the database import, it's mandatory to define these mappings to ensure data integrity and accuracy.

Lock and Claim System: Managing Concurrent Imports

A unique aspect of the CSV Import feature is its lock and claim system, designed to maintain order and prevent conflicts during the import process. In essence, this system ensures that only one import operation can occur at any given time across the entire application instance. When a user initiates an import, the feature is locked, rendering it inaccessible to other users until the operation is either rolled back, resolved, or explicitly released by the initiating user.



This lock mechanism is particularly crucial in scenarios where the importing process is interrupted—such as a browser closure or an unexpected exit from the platform. Upon returning, the user who initiated the import will encounter a warning message prompting them to reclaim the import process. This action can only be performed by the original user or an authorized user, such as an administrator, highlighting the importance of proper access controls and user permissions in managing data imports.



Through these steps, the CSV Import feature empowers you to efficiently populate your entities with external data, enhancing your application's data management capabilities and overall utility.

Edit this page

Last updated by ***Miqueas***

Previous
« Data Elements

Next
CSV Export »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Data Model
Data Elements
CSV Import
CSV Export
Access Control
Login
Advanced

CSV Export

Entity Selection and Export Configuration
Preview and Log
Finalizing the Export

The CSV Export functionality mirrors the CSV Import feature in its structural setup, offering a seamless way to extract data from your application into CSV files. This feature facilitates data sharing, analysis, and backup, ensuring your data is accessible and manageable outside the platform.

Entity Selection and Export Configuration

The process begins in the sidebar, where you select the entity whose data you wish to export. This initial step ensures that the data for export is precisely targeted, aligning with your specific needs.

EntitySelect

Upon selecting an entity, the content area presents you with options to customize your export. On the left side, you can define the export settings.

Settings

- **Encoding Format**
  Select the encoding (UTF-8, ASCII, etc.) for your CSV file. This choice guarantees that the exported file will be compatible with the systems and applications you intend to use it with.
- **Header Row Inclusion**
  Decide whether to include a header row in the CSV file. This row contains the names of the fields from your entity, providing clarity and structure to the exported data.
- **Separator Character**
  Choose the character (comma, semicolon, etc.) used to separate columns in the resulting CSV file. This setting ensures that the exported data aligns with your preferred CSV format.

Preview and Log

To the right of the export settings, a "preview" section provides a glimpse of how the exported CSV file will appear based on your chosen settings. This preview, displaying a portion of the data within the chosen entity, allows you to verify the format and content before proceeding with the export.

Preview:

#	NewProperty	NewProperty1	NewProperty2	NewProperty3	NewProperty4	NewProperty5	NewProperty6
0	42	1971-01-18	42	0	0	0	0
1	4711	2000-01-01	4711	0	0	0	0

Above the content area, the header hosts the export button.

HeaderExport

Initiating the export process generates a new entry in the log, located at the bottom of the content area. This log section updates you with messages about the export process, including successful completion notifications or any issues that may arise.

Log

14.3.2024 15:26:01: New export session started.
14.3.2024 15:26:01: Well done! No existing exportId was found.
14.3.2024 15:28:24: ------- New export ---------
14.3.2024 15:28:24: current progress: 100%
14.3.2024 15:28:24: SUCCESS CSV was successfully imported into the database.

Finalizing the Export

Once the export is completed, you are prompted to either download the CSV file or "resolve" the operation. This step is crucial for managing the CSV Export feature's lock mechanism, which, similar to the CSV Import, prevents simultaneous export operations. The system remains locked until the export is either downloaded or resolved, ensuring controlled access and preventing data conflicts.

HeaderClear

Through its intuitive interface and comprehensive options, the CSV Export feature offers a powerful tool for exporting data from your application. By allowing precise control over the export settings and providing a secure environment for data extraction, it ensures that your application's data remains accessible, portable, and ready for use in various contexts.

Edit this page

*Last updated by **Miqueas***

Previous
« CSV Import

Next
Access Control »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Access Control
Roles
Users
Login
Advanced

Managing Users and Roles
Two Types of Sign-In
Users
Roles

Access Control

Access Control is divided into two main functionalities: managing users and roles within your application, and managing access for developers within the yeet platform. This dual functionality ensures both operational security for your application and organizational control for your development team.

Managing Users and Roles

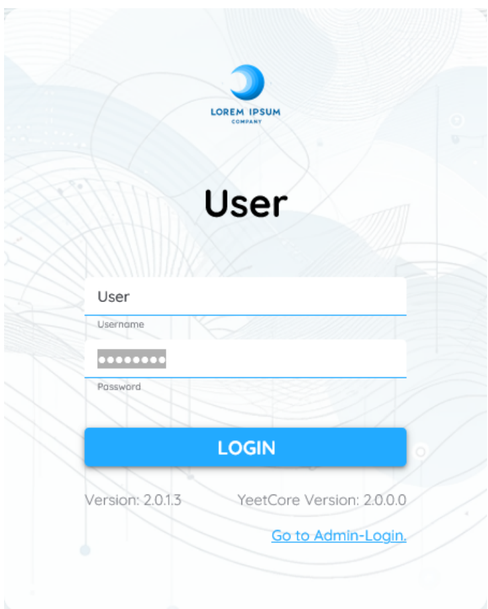
Access Control enables you to create and manage roles, defining the scope of permissions for various users within your application. These roles range from admin positions with comprehensive access to more restricted roles for end-users, ensuring a clear structure of permissions and access. By assigning users to specific roles, you tailor their experience and access rights within the application, enhancing both security and user experience.

- **Create and Manage Roles**
  Define roles with specific permissions tailored to the needs of your application and team.
- **Assign Users to Roles**
  Allocate roles to application users and developers, streamlining access control based on their responsibilities.

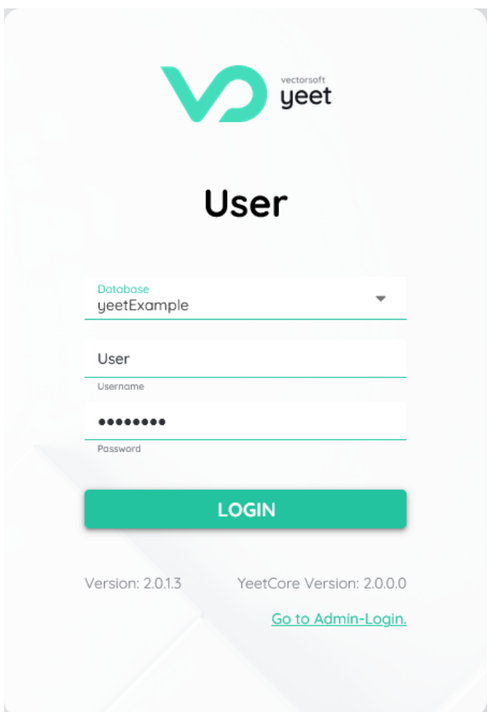
Two Types of Sign-In

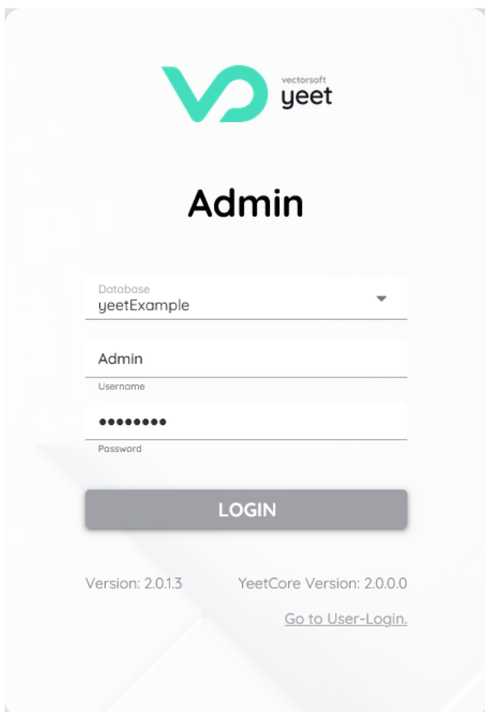
Access Control supports two different types of sign-in methods to cater to both users of your application and developers on the yeet platform:

- **Application User Sign-In**
  This sign-in method is designed for users of your application. It allows them to access and interact with the app according to the permissions set by their assigned role. This is crucial for operational security and user experience, ensuring users can only access what they are authorized to.



- **Developer Sign-In**
  This method is reserved for developers working within the yeet platform. It provides access to development tools, resources, and the ability to manage the application's backend, including user and role management. Developer sign-in ensures that only authorized personnel can make changes to the application, protecting the integrity of your project.





Users

The Users section allows for the creation and management of user accounts within your application. Here, you can add new users, edit existing user details, and assign roles to each user. Assigning roles is a vital part of user management, as it determines the access level and permissions each user has within your application. Detailed explanations and instructions for managing users can be found on the dedicated Users page of our documentation.

accessControl
User

Roles

In the Roles section, you have the ability to define and customize roles, which are essentially collections of permissions. Roles enable you to specify default rights for a certain type of access and to configure specific rights for particular types of data or functions. By defining roles, you can easily control what actions users can perform, which parts of the application they can access, and what data they can view or modify. Each role can be tailored to fit the needs of different user groups within your application, from administrators with full access to specific roles with limited permissions aimed at standard users or external collaborators. For more detailed information on creating and managing roles, along with setting up rights and permissions, please refer to the dedicated Roles page in our documentation. Integrating Users and Roles

accessControl
Role

Edit this page

*Last updated by **Felix***

Previous
« CSV Export

Next
Roles »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Access Control
Roles
Users
Login
Advanced

Sidebar and Role Creation
Editing Role Properties
Default Rights
Specific Rights
Custom Rights Picker

Roles

In our platform, roles play a crucial part in defining the access level and permissions for users within your application. The Roles section provides a comprehensive interface for managing these roles, from creation to customization of rights associated with each role.

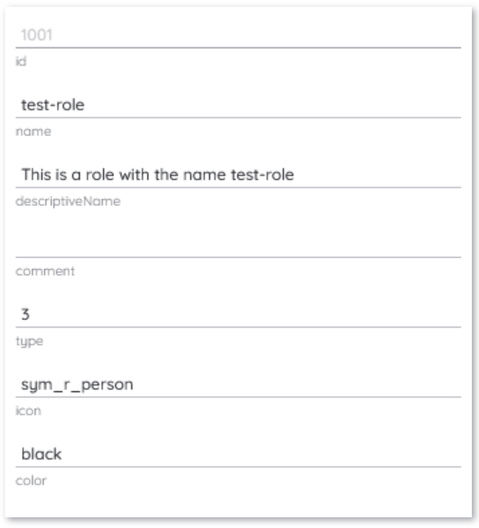
Sidebar and Role Creation

On the left side of the Roles page, you will find a sidebar listing all the roles you have created. At the bottom of this sidebar, a button allows you to add a new role. Clicking this button prompts a dialog where you can specify a name for your new role. Once a role is selected from the sidebar, it becomes active, enabling you to edit its properties and rights within the content area of the page.

An important feature to highlight within the role management interface is the filtering capability provided in the topbar. This feature allows you to filter rights based on the presetTypes such as "entity," "enum," "action," and any custom types you have defined. Utilizing this filter can significantly streamline the process of managing rights by enabling you to focus on the specific types of interest at any given time.

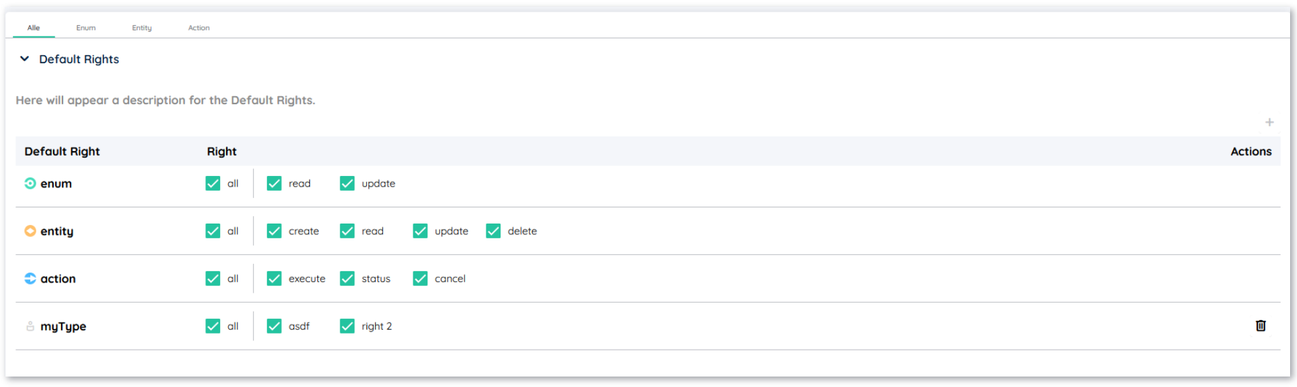
Editing Role Properties

When a role is activated, the content area provides options to edit both defaultRights and specificRights associated with that role. Additionally, a sidebar on the right side of the content area is available for editing meta information about the role, such as its name, icon, and icon color. All changes made in these sections are saved instantly, including modifications to rights and meta information.



Default Rights

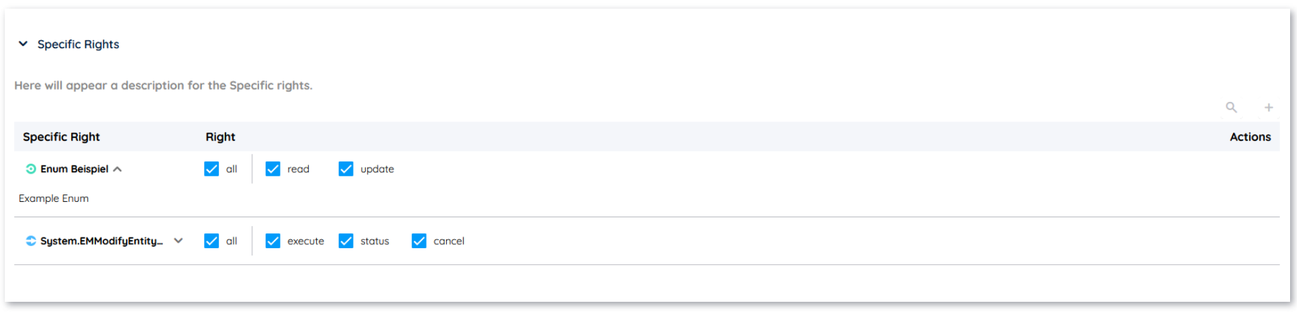
Default rights define standard permissions for a preset type like "entity," "enum," and "action." Each default right details the standard behavior for actions like reading, writing, and deleting.



> **SECURITY NOTICE: ACCESS PERMISSIONS**
>
> A crucial aspect to be aware of is the platform's default behavior towards unspecified rights: they are initially set to true. This setting is vital for maintaining accessibility and functionality but requires careful adjustment to ensure it aligns with your intended access controls. It's essential to review and modify these settings as necessary to prevent unintended access, ensuring secure and effective role-based access control.

Specific Rights

Specific rights allow for fine-grained control over permissions for specific objects within your application. For preset types "entity" and "enum," the selection list is linked to the actual objects you've created in the Data section, ensuring consistency across your application. The "action" category includes all server-side actions defined by you or pre-provided by the platform.

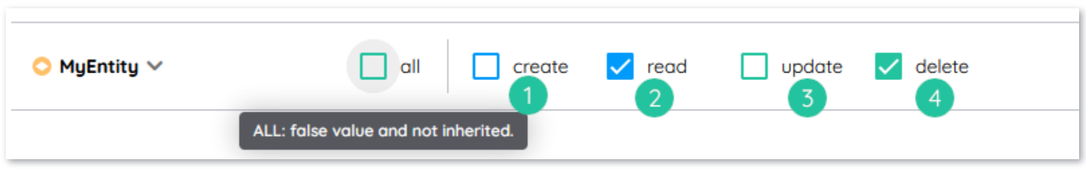


For custom types, you can specify the names for right objects yourself. It is essential to maintain consistency in naming these right objects across different roles manually.

If a specific right for an object that does not exist is queried, the platform will revert to the value specified in the default right for that type. Note that specific rights can only be created if the corresponding default right for the type is defined.

Custom Rights Picker

The picker for custom rights introduces states: true-inherited, false-inherited, true-not-inherited, and false-not-inherited. The inherited states are defined by the defaultRights. Clicking the checkbox in an inherited state changes it to false-not inherited, and another click changes it to true-not inherited. Clicking again retrieves the inherited status from the corresponding default right. A blue frame or checkmark indicates an inherited state, while green signifies not inherited.



1. false-inherited
2. true-inherited
3. false-not-inherited
4. true-not-inherited

Edit this page

*Last updated by **Wegener***

Previous
« Access Control

Next
Users »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Access Control
Roles
Users
Login
Advanced

Tile View
List View

Users

Welcome to the User Management section of our platform, where you can oversee and manage user accounts, including their roles, access permissions, and personal information. This guide provides a comprehensive overview of the features available in the User Management interface, including navigating between different views, adding new users, and managing user details.

Navigating the User Interface

Upon accessing the User Management page, you'll notice the header menu offers options to switch between a Tile View and a List View, in addition to search functionality and the option to add a new user component. When clicking the search icon a search field can be toggled on and off.

usersHeader

Tile View

In the Tile View, users are represented by tiles that display significant information at a glance:

- **Role Color and Icon**
  Each tile includes the role's assigned color and icon at the top, providing an easy visual cue to identify the user's role within the platform. {" "}

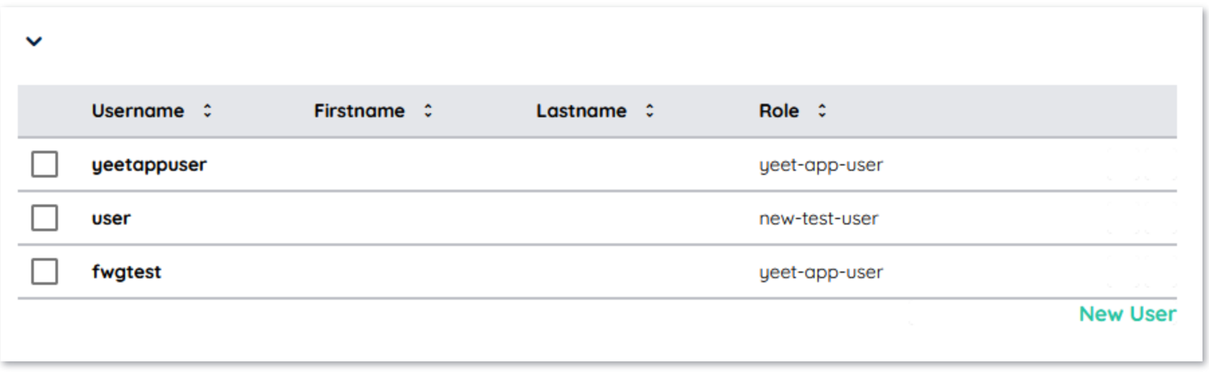
usersTile

- **Edit and Delete Options**
  Each user tile is equipped with a gear icon for editing and a trash bin icon for deleting the user. Clicking on the role itself or the gear icon opens the edit dialog, while selecting the trash bin icon prompts the user deletion dialog. {" "}

usersDeleteDialog

List View

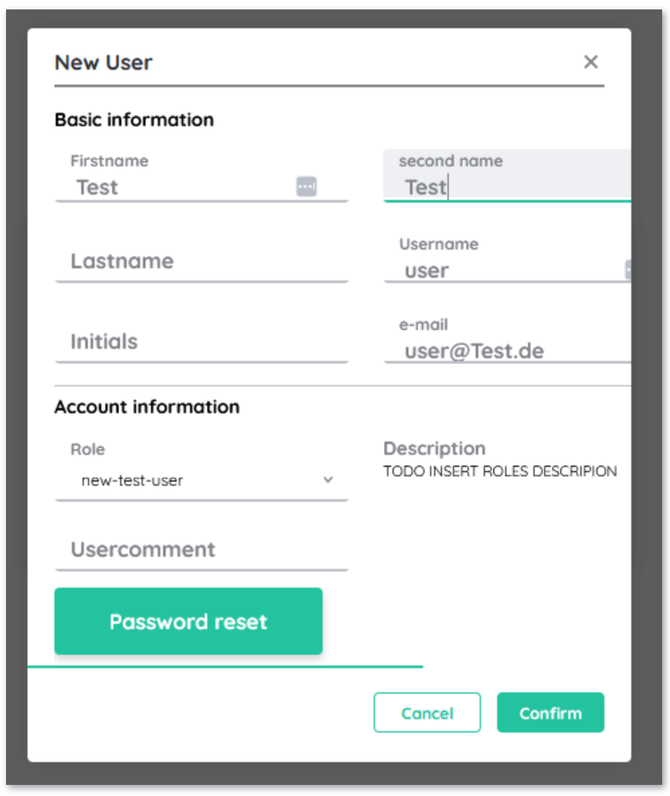
The List View organizes users in a tabular format, maintaining the same functionalities as the Tile View but in a list:



- **Role Color and Icon**
  Similar to the Tile View, the role's color and icon are displayed on the left side of each list item.
- **Action Buttons**
  The edit (gear icon) and delete (trash bin icon) action buttons are located on the right side of the list entry. Clicking on the list entry itself or the gear icon will open the edit dialog for the user.

User Edit Dialog

When entering the edit dialog for a user, you'll have the ability to modify various properties e.g. including name, email address and role assignment.



Additionally, this dialog provides a Password Reset Button. Utilizing this button will reset the user's password to a temporary state. Upon their next login attempt, the user will be prompted to set a new password, enhancing security and ensuring that only the authorized user has access to the account.

Adding a New User

To add a new user to the system, select the option to add a new component from the header menu. Alternatively in list view the "New User" label can be clicked in the bottom. This action will guide you through the process of specifying the new user's details, including their name, email address, and role within the platform. Following the submission of this information, the new user will be added to the User Management page and can be viewed in either the Tile or List View.

Managing Users

The search functionality in the header menu allows for quick filtering of users based on keywords, making it easier to locate specific accounts. Whether you need to update user details, reset passwords, or manage roles, the User Management section provides a centralized interface for all your user administration needs. In list view the entries can be filtered by clicking the column that you want to sort by. By clicking once you choose ascending sorting, by clicking again you will choose descending sorting.

usersHeaderSearch

Edit this page

*Last updated by **Wegener***

Previous
« Roles

Next
Login »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Cockpit
Themes
Projects
Frontend
Data
Access Control
Roles
Users
Login
Advanced

Tile View
List View

Users

Welcome to the User Management section of our platform, where you can oversee and manage user accounts, including their roles, access permissions, and personal information. This guide provides a comprehensive overview of the features available in the User Management interface, including navigating between different views, adding new users, and managing user details.

Navigating the User Interface

Upon accessing the User Management page, you'll notice the header menu offers options to switch between a Tile View and a List View, in addition to search functionality and the option to add a new user component. When clicking the search icon a search field can be toggled on and off.

usersHeader

Tile View

In the Tile View, users are represented by tiles that display significant information at a glance:

- **Role Color and Icon**
  Each tile includes the role's assigned color and icon at the top, providing an easy visual cue to identify the user's role within the platform. {" "}

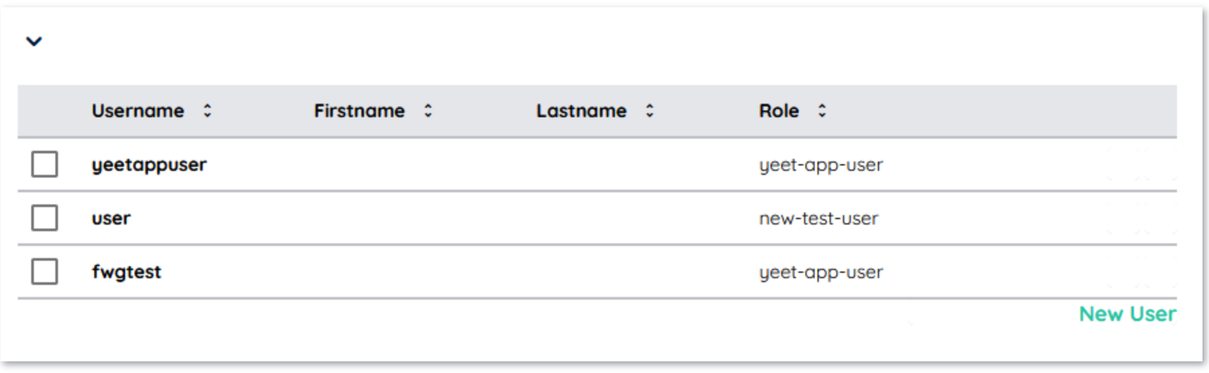
usersTile

- **Edit and Delete Options**
  Each user tile is equipped with a gear icon for editing and a trash bin icon for deleting the user. Clicking on the role itself or the gear icon opens the edit dialog, while selecting the trash bin icon prompts the user deletion dialog. {" "}

usersDeleteDialog

List View

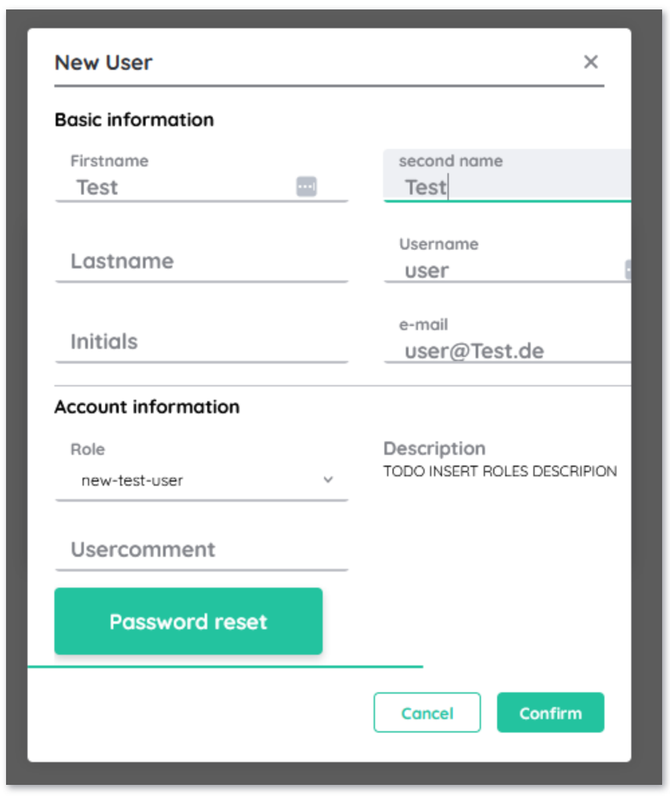
The List View organizes users in a tabular format, maintaining the same functionalities as the Tile View but in a list:



- **Role Color and Icon**
  Similar to the Tile View, the role's color and icon are displayed on the left side of each list item.
- **Action Buttons**
  The edit (gear icon) and delete (trash bin icon) action buttons are located on the right side of the list entry. Clicking on the list entry itself or the gear icon will open the edit dialog for the user.

User Edit Dialog

When entering the edit dialog for a user, you'll have the ability to modify various properties e.g. including name, email address and role assignment.



Additionally, this dialog provides a Password Reset Button. Utilizing this button will reset the user's password to a temporary state. Upon their next login attempt, the user will be prompted to set a new password, enhancing security and ensuring that only the authorized user has access to the account.

Adding a New User

To add a new user to the system, select the option to add a new component from the header menu. Alternatively in list view the "New User" label can be clicked in the bottom. This action will guide you through the process of specifying the new user's details, including their name, email address, and role within the platform. Following the submission of this information, the new user will be added to the User Management page and can be viewed in either the Tile or List View.

Managing Users

The search functionality in the header menu allows for quick filtering of users based on keywords, making it easier to locate specific accounts. Whether you need to update user details, reset passwords, or manage roles, the User Management section provides a centralized interface for all your user administration needs. In list view the entries can be filtered by clicking the column that you want to sort by. By clicking once you choose ascending sorting, by clicking again you will choose descending sorting.

usersHeaderSearch

Edit this page

*Last updated by **Wegener***

Previous
« Roles

Next
Login »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yee?
Getting Started
UI Components
Essentials
yee! Platform
Advanced
Components Basics
Basic Properties
Multi Picker
ODatra

Components Basics

Here are some basic properties that are **common** across **all components**. Here, we'll introduce each property briefly. More detailed information about properties specific to **individual components** can be found on their respective **pages**.

Basic Structure of Components

Each component behaves like a flex item in a flex container. Allowing you to easily create a responsive UI layout. For more information on the layout system, **click here**.

Furthermore, each component has the following basic structure:

- A component can have one or more states (normal, hover, active, focus, disabled, error, readonly).
- Each component has a background color for each state.
- Each component has a border for each state.
- Each component can maintain self-defined spacings to its content or to other components. Refer to the **Spacings section** for more details.

Basic Properties

The following properties are common to every component.

Generic | Style

- META
- DISPLAY
- SIZE
- PLACEMENT

Generic

Generic > Meta

Only abstract properties, that won't have a direct influence on the look and structure of the component, occur in the meta-group. Properties from this group can only be edited via **Detail Panel**.

componentId

The componentID is the identifier of the component, which is unique inside a page. The **componentID** is used for: `function/Property`

- Addressing components in functions
- Modifying the component's configuration via JavaScript
- Linking event functions with a component

> **INFO**
> This property is read-only and is assigned by the system.

componentType

The componentType is the type of the component. With the help of the component type and the component id, a component can be uniquely addressed. The component type is mostly used to provide more information to verify that the correct element has been addressed. In the Components-Section inside the documentation's sidebar you will find a list of all possible component types.

> **INFO**
> This property is read-only and is assigned by the system.

name

The name is the custom name of the component, which is used for better identification of the component. `string/Property`

version

⚠ warning This property is no longer needed and will be deprecated in the future. ⚠

coreTheme

Themes are associated with a CoreTheme. This allows a CoreTheme to include multiple SubThemes as well as additional theme data. To apply a theme, the CoreTheme must be selected first, followed by an associated SubTheme. For more information on themes, please refer to **themes**. `string/Property`

subTheme

When a CoreTheme is selected, the applicable SubTheme containing the component styles is chosen here. `string/Property`

groupTheme

The theme group contains specific component variations within the SubTheme. `string/Property`

> **DANGER**
> This functionality has not been integrated yet.

Generic > Display

In the display group properties occur that will influence the way the component is displayed and arranged with its neighboring components.

display

This property specifies the display behavior of the component. This can be set to: `List/Property`

`none block flex inline`

Some of the display property's specifications are set in different properties according to the selected value of the display property. For example, when flex is selected further specification can be done in the flex property.

position

This property specifies the type of positioning method used for the component. Default will be `static`, unless/until a position is always specified by top, bottom, left or right values. Depending on the selected value for the position, the starting point for those values will differ. `List/Property`

> **INFO**
> In the future, additional options such as relative, absolute, sticky, and fixed will also be made available.

visible

This property can toggle the visibility of the component. There are three options: `List/Property`

- **inherit**
  The visibility is inherited from an outer component.
- **hidden**
  The component is invisible.
- **visible**
  The component is visible.

This will only hide the element, however if you want to prevent the element from loading you should set the display property to "none".

Generic > Size

sizing

In the sizing group properties define the size of the yComponent. You can either set hard values for size, or alternatively rely on the flex box system, where you will only have to set the min- and max-sizes. PBI_SizingPicker

These properties allow for precise control over the dimensions of the component, including setting constraints on its maximum and minimum sizes.

- **width:**
  Specifies the width of the component.
- **maxWidth:**
  Sets the maximum width that the component can have.
- **minWidth:**
  Defines the minimum width that the component can have.
- **height:**
  Specifies the height of the component.
- **maxHeight:**
  Sets the maximum height that the component can have.
- **minHeight:**
  Defines the minimum height that the component can have.

flex

The flex property sets the length on flexible items. This sets the value in n-fold shares extrapolated to all other components with flex-display within the same container. This property's functionality works well together with setting a max-width and max-height to set an upper limit in size or min-width and min-height to set a lower limit in size. `string/Property`

Generic > Placement

In the placement group properties can be adjusted that influence the distances to neighboring elements outside and included elements inside. PBI_SpacingPicker

> **INFO**
> For more info see Box/Model and Placement basics.

margin

These properties allow you to define the spacing between the edges of the component and its surrounding elements in each direction.

The list summarizes properties related to the margin:

- **marginTop:**
  Specifies the margin at the top of the component.
- **marginRight:**
  Specifies the margin on the right side of the component.
- **marginBottom:**
  Specifies the margin at the bottom of the component.
- **marginLeft:**
  Specifies the margin on the left side of the component.

padding

These properties allow you to define the spacing between the content of the component and its border in each direction.

The list summarizes properties related to the padding:

- **paddingTop:**
  Specifies the padding at the top of the component.
- **paddingRight:**
  Specifies the padding on the right side of the component.
- **paddingBottom:**
  Specifies the padding at the bottom of the component.
- **paddingLeft:**
  Specifies the padding on the left side of the component.

Style

The style category contains properties that configure the look and feel of the component. Some of these properties are the same for all components, some are added specifically for a certain component. In this case only the style properties that are general should be explained.

Style > Background

In this background group the appearance of the background can be adjusted. For most components only the bgColor property can be found in this group.

bgColor

The bgColor is the color of the background. This can be set as a color from a palette or a custom hex color. Each component state has its own background color. PBI_ColorPicker `Color/Property`

Style > Border

Properties that will change the appearance of the components border can be found in the border group. Some properties are responsible for only one of the borders in the top, bottom, left or right. Other properties are responsible for all four of them at once. Toolbar_BorderExtension

color

Here, a separate color can be set for each edge. The colors can also be chosen differently for each state.

Property names for each edge:

- **borderTopColor:**
  Color of the top border
- **borderRightColor:**
  Color of the right border
- **borderBottomColor:**
  Color of the bottom border
- **borderLeftColor:**
  Color of the left border

style

Border style defines the style of the border of an element, such as `solid` `dashed` or `dotted`.

Property names for each edge:

- **borderTopStyle**
  Style of the top border
- **borderRightStyle**
  Style of the right border
- **borderBottomStyle**
  Style of the bottom border
- **borderLeftStyle**
  Style of the left border

width

Specifies the width of the border of an element.

Property names for each edge:

- **borderTopWidth**
  width of the top border
- **borderRightWidth**
  width of the right border
- **borderBottomWidth**
  width of the bottom border
- **borderLeftWidth**
  width of the left border

borderRadius

Defines the rounding of the corners. Applies to all states and edges.

Style > Shadow

Properties concerning the appearance of the shadow around the component can be adjusted in the shadow group. For most components only the box shadow property can be found in this group.

For this example shadow the settings were: x=10 y=20 blur=0 spread=0. The x and y values will move the whole shadow. For positive x values the shadow will be moved to the right, for negative ones to the left. For positive y values the shadow will be moved to the bottom, for negative ones to the top. The unit of the values is pixels in all cases.

For this example shadow the settings were: x=10 y=20 blur=5 spread=0. The blur value is also given in pixels and will blur the shadow depending on the value, the higher the value the more intense the blur. The value has to be positive, negative values are forbidden.

For this example shadow the settings were: x=10 y=20 blur=0 spread=30. The spread value is also given in pixels. This value is responsible for making the shadow, which by default is the same width and height as its component bigger or smaller. A positive value will increase the width and height of the shadow a negative value will decrease it.

PBI_ShadowPicker

boxShadow

The shadow of the component. This is set by the four sub-properties x, y, blur and spread of the shadow property, which can be further read about in the shadow property section. `string/Property`

Style > Opacity

In the opacity group you can find properties concerning the opacities of areas belonging to the component. For most components only the opacity property can be found in this group.

opacity

The opacity of all colored parts of the component. This value is set in percent. `number/Property`

Events

Everything related to events can be set in this section. The custom function calls and remote procedure calls are also made in this section.

Events > Lifecycle

This is where you can manage the events associated with the lifecycle of the component.

Evtinit

This section allows you to access the editor for defining the component's init function. Within this editor, you can write custom JavaScript code, which gets executed whenever the component is fully mounted.

Basic Structure of Components
Basic Properties
Generic
Generic > Meta
Generic > Display
Generic > Size
Generic > Placement
Style
Style > Background
Style > Border
Style > Shadow
Style > Opacity
Events
Events > Lifecycle

Copyright © 2024 YeeDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
OData

Comparison Operators
Logical Operators
String Functions
Arithmetic Operators
Date Functions

OData

In the world of OData (Open Data Protocol), efficiently querying and interacting with data services is a key capability. This guide will introduce you to the various OData column (property) types and the filter operators that can be applied to them for effective data querying.

OData Column Types

OData services support a range of data types for properties, each designed to accommodate specific kinds of data:

- **Edm.String**
  Represents text data. It can contain letters, numbers, and other characters.
- **Edm.Int16, Edm.Int32, Edm.Int64**
  Integer data types for representing whole numbers of varying sizes.
- **Edm.Decimal**
  Used for decimal numbers, providing a high level of precision.
- **Edm.Double**
  Represents floating-point numbers.
- **Edm.Boolean**
  Represents boolean values (true or false).
- **Edm.Date**
  Used for dates without time information.
- **Edm.TimeOfDay**
  Represents time without date information.
- **Edm.DateTimeOffset**
  Represents date and time with timezone information.

Filter Operators

OData's powerful query options include a variety of filter operators that can be applied to entity properties in a query. These operators allow for complex filtering based on the property values:

Comparison Operators

- **eq (equals)**
  Checks if the value of a property is equal to a specified value.
- **ne (not equals)**
  Checks if the value of a property is not equal to a specified value.
- **gt (greater than), ge (greater than or equal to)**
  These operators are used for comparing numerical values, including dates and times.
- **lt (less than), le (less than or equal to)**
  Similar to gt and ge, these operators compare numerical values.

Logical Operators

- **and**
  Combines two or more conditions that must all be true for the filter to pass.
- **or**
  Combines two or more conditions where at least one must be true for the filter to pass.
- **not**
  Negates the condition that follows.

String Functions

- **startswith, endswith**
  Check if a string property starts or ends with a specified substring.
- **contains**
  Checks if a string property contains a specified substring.
- **substringof (deprecated in OData version 4.0)**
  Similar to contains, checks if a property string contains a specific substring.

Arithmetic Operators

- **add, sub, mul, div, mod**
  Perform basic arithmetic operations on numerical properties.

Date Functions

- **year, month, day**
  Extracts the year, month, or day component from a Date or DateTimeOffset property.

Usage in Queries

This will be mostly useful when using custom queries in javascript within UILogic. Filter operators can be used in the $filter query option to specify conditions that entities must meet to be included in the response. Here's a simple example:

```
http://example.com/odata/People?$filter=FirstName eq 'John' and Age gt 30
```

This query retrieves people named "John" who are older than 30.

It's important to note that the availability of certain filter functions and operators may vary depending on the OData service implementation. Always refer to the specific documentation for the OData service you are querying.

By leveraging these column types and filter operators, users can construct powerful queries to efficiently access and manipulate data in an OData service.

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
Sizing
Spacing
Border
Alignment
Font
Element
OData

Sizing

Description
Dimension Properties
Occurences
Toolbar
Detail Panel

Description

The sizing group can be found either in Toolbar (e.g. Sizing Picker) or Detail Panel. It groups any sizing related property.

In the sizing group properties define the size of the yComponent. You can either set hard values for size, or alternatively rely on the Flexbox system, where you will only have to set the min- and max-sizes.

Dimension Properties

minWidth

The minimum value for the width of the component. This can be set in: UnitNumberProperty

px pt em vw vh %

This property can be used in combination with the display set to "flex", since then the width will grow according to the Flexbox system, while not being able to go lower than the specified min-width.

minHeight

The minimum value for the height of the component. This can be set in: UnitNumberProperty

px pt em vw vh %

This property can be used in combination with the display set to "flex", since then the height will grow according to the Flexbox system, while not being able to go lower than the specified min-height.

width

The value for the width of the component. This can be set in: UnitNumberProperty

px pt em vw vh % `auto`

In combination with the display set to "flex", this property's unit should be set to "auto". Otherwise the width will not be adjusted dynamically according to the Flexbox system.

height

The value for the height of the component. This can be set in: UnitNumberProperty

px pt em vw vh % `auto`

In combination with the display set to "flex", this property's unit should be set to "auto". Otherwise the height will not be adjusted dynamically according to the Flexbox system.

maxWidth

The maximum value for the width of the component. This can be set in: UnitNumberProperty

px pt em vw vh % `none`

This property can be used in combination with the display set to "flex", since then the width will grow according to the Flexbox system, while not being able to go higher than the specified max-width.

maxHeight

The maximum value for the height of the component. This can be set in: UnitNumberProperty

px pt em vw vh % `none`

This property can be used in combination with the display set to "flex", since then the height will grow according to the Flexbox system, while not being able to go higher than the specified max-height.

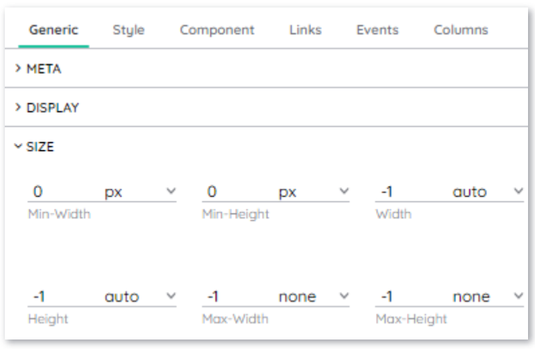
Occurences

Toolbar

In the Toolbar the sizing can be adjusted. Here you can change sizing related options.

Detail Panel

Here the sizing can also be adjusted. It doesn't differ from the similar picker inside the Toolbar.



Edit this page

*Last updated by **Felix***

Previous
« Multi Picker

Next
Spacing »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
Sizing
Spacing
Border
Alignment
Font
Element
OData

Description
Margin
Padding
Occurences
Toolbar
Detail Panel

Spacing

Description

The spacing group can be found either in Toolbar or Detail Panel. It groups any spacing related property.

Margin

margin-top

The space is created outside of the top border. This can be set in: UnitNumberProperty

% or px

margin-right

The space is created outside of the right border. This can be set in: UnitNumberProperty

% or px

margin-bottom

The space is created outside of the bottom border. This can be set in: UnitNumberProperty

% or px

margin-left

The space is created outside of the left border. This can be set in: UnitNumberProperty

% or px

Padding

padding-top

The space is created inside of the top border. This can be set in: UnitNumberProperty

% or px

padding-right

The space is created inside of the right border. This can be set in: UnitNumberProperty

% or px

padding-bottom

The space is created inside of the bottom border. This can be set in: UnitNumberProperty

% or px

padding-left

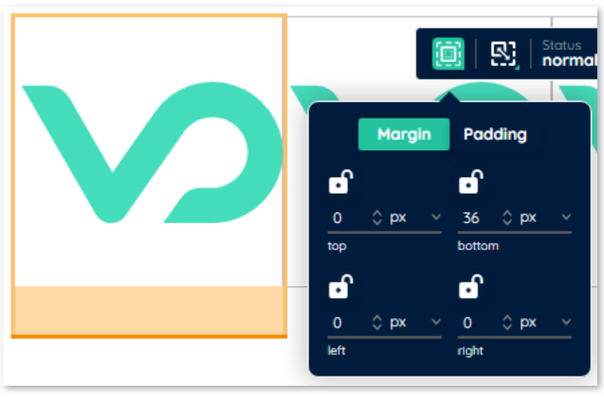
The space is created inside of the top border. This can be set in: UnitNumberProperty

% or px

Occurences

Toolbar

Inside the Toolbar, the spacing can be changed via the Spacingbar. It's a simplified version which is similar to the Sizing Picker.



The adjustment can be made one by one for each side. Furthermore, with aid of the lock, mutliple sides can be locked to apply changes to every locked side. This simplifies changes which for example should be applied to all sides.

Moreover the Spacingbar is supplemented by the **InPlSpacing**.
This tool allows the adjustment of spacing related properties via drag and drop. When activating the Spacingbar, the **InPlSpacing** also gets activated. The borders of each side can be dragged to change the spacing, which can be done for margin and also for padding.

The lock possibility also applies for this special tool. This means, whenever multiple sides are locked, dragging any border will result in dragging all locked border equally, while only dragging one border.

Detail Panel

Here the picker differs from the Spacingbar and offers an alternative presentation, that is inspired by the classic spacing depiction of web browsers.

Essentially, the picker offers a slightly less extensive variety of possibilities. For example, the lockability doesn't apply to this picker and and also the adjustment via drag and drop is not offered. Therefore this picker is a minified version which is reduced to the necessary basics.
The depiction can be seperated into three compartments. In the center area, the dashed square represents the actual component itself. The surrounding ring depicts the padding, while the outer ring adjusts the margin.

Edit this page

*Last updated by **Felix***

Previous
« Sizing

Next
Border »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
Sizing
Spacing
Border
Alignment
Font
Element
OData

Border

Description
Border Color
Border Style
Border Width
Border Radius
Occurences
Toolbar
Detail Panel

Description

The border group can be found either in Toolbar (e.g. Border Picker) or Detail Panel. It groups any border related property.

Border Color

border-top-color

The color of the top border. This can be set as a color from a palette or a custom hex color. ColorProperty

border-right-color

The color of the right border. This can be set as a color from a palette or a custom hex color. ColorProperty

border-bottom-color

The color of the bottom border. This can be set as a color from a palette or a custom hex color. ColorProperty

border-left-color

The color of the left border. This can be set as a color from a palette or a custom hex color. ColorProperty

Border Style

border-top-style

The style of the top border. This can be set to: ListProperty

`solid` `dashed` `dotted` `none`

border-right-style

The style of the right border. This can be set to: ListProperty

`solid` `dashed` `dotted` `none`

border-bottom-style

The style of the bottom border. This can be set to: ListProperty

`solid` `dashed` `dotted` `none`

border-left-style

The style of the left border. This can be set to: ListProperty

`solid` `dashed` `dotted` `none`

Border Width

border-top-width

The width of the top border in px. UnitNumberProperty

border-right-width

The width of the right border in px. UnitNumberProperty

border-bottom-width

The width of the bottom border in px. UnitNumberProperty

border-left-width

The width of the left border in px. UnitNumberProperty

Border Radius

border-radius

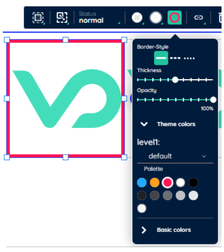
The border radius defines radius of the corners of all the borders. This can be set in % or px. UnitNumberProperty

Setting the border radius in percentage can help if you, for example, want to create a circle. In this case you could set each border radius to 50 % while setting width and height of the component to the same value.

Occurences

Toolbar

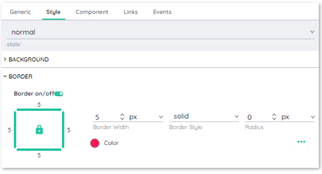
Here you can adjust common border related properties related to an state. It basically depicts a simplified version of the related picker inside Detail Panel.



Besides adjusting the common styling, choosing the color is special. A Color Picker is included in the process and allows choosing colors based on the active SubTheme or from basic colors.

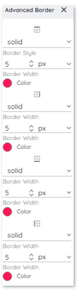
Detail Panel

Inside the Detail Panel a picker for adjusting any border related properties can be found, which extends the possibilites of it's relative in Toolbar.



In comparison to it's relative picker inside Toolbar, this picker differs. Here, the border must be toggled to be changeable. This allows to easily turn off the border, if no border is needed.
The depiction on the left side of the picker symbolizes the component itself. By clicking on the edges, you can select and deselect the border for hiding single borders.

The big difference in this picker is that you can adjust every border individually to fit your expectations and style guide. This means, that the top border can differ from it's equivalent on the bottom. To achieve similar results, you can toggle a dialog on the right side of the picker, by clicking on the three dots.



Inside the dialog you can adjust any styling related property for every single border indivually and achieve a result of your liking. In comparison to the color picker inside the Borderbar you can also choose custom colors.

Edit this page

*Last updated by **Felix***

Previous
« Spacing

Next
Alignment »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
Sizing
Spacing
Border
Alignment
Font
Element
OData

Description
Alignment Properties
Occurences
Toolbar
Detail Panel

Alignment

Description

The alignment group can be found either in Toolbar or Detail Panel. It groups any alignment related property.

Alignment Properties

flex-wrap

This sets whether the elements inside the component are forced onto one line or can wrap onto multiple lines. BooleanProperty

This can be either further configured in the Detail Panel or via the dedicated **Flex-Wrap** Picker that can be found in Toolbar by **choosing between horizontal and vertical**.

flex-direction

This sets how the elements inside the component are placed in the flex container defining the main axis and the direction. It can be set in: ListProperty

`row` `column` `row-reverse` `column-reverse`

The properties **flexDirection**, **justifyContent** and **alignItems** can be either further configured in the Detail Panel or via the dedicated **Flex-Direction** Picker that can be found in Toolbar by **choosing between horizontal and vertical**.

justify-content

This sets how the elements inside the component are placed along the main axis. It can be set in: ListProperty

`start` `center` `end` `space-between` `space-around`

align-items

This sets how the elements inside the component are placed along the cross axis. It can be set in: ListProperty

`start` `center` `end` `baseline` `stretch`

justify-items

This sets how the elements inside the component are justified along the appropriate axis. It can be set in: ListProperty

`start` `center` `end` `baseline` `stretch`

align-content

This sets how the elements inside the component are placed along the cross axis. It doesn't affect single-line containers. This can be set in: ListProperty

`start` `center` `end` `space-between` `space-around` `stretch`

overflow-x

This sets the behavior for content if it doesn't fit inside the container in horizontal direction. It can be set in: ListProperty

`visible` `hidden` `scroll` `auto`

overflow-y

This sets the behavior for content if it doesn't fit inside the container in vertical direction. It can be set in: ListProperty

`visible` `hidden` `scroll` `auto`

show-scrollbar

This controls whether a scrollbar is displayed or not, while scrolling is still allowed. BooleanProperty

Occurences

Toolbar

A minified selection of the most common properties related to the alignment is found here.

@Note insert image here

The dedicated picker is separated into three different parts. First of all you select a flex direction via the Toolbar which will open another Toolbar Extension to adjust further properties related to it. In addition to that you can toggle flex wrap via the bar seperately.

Detail Panel

Inside the Detail Panel a more detailed adjustment is presented. This picker offers the ability to manipulate each property related to the alignment individually.

@Note insert image here

Edit this page

*Last updated by **Wegener***

Previous
« Border

Next
Font »

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
Sizing
Spacing
Border
Alignment
Font
Element
OData

Font

Description
Font Properties
Occurences
Toolbar
Detail Panel

Description

The font group can be found either in the Toolbar or the Detail Panel. It groups any font related property.

Font Properties

font-color

The color of the text. This can be set as a color from a palette or a custom hex color. You can choose a color via the Color Picker.

ColorProperty

font-family

The font family of the text. This can be set to a font family from the specified list of fonts. The preselected font family is "Quicksand".

ListProperty

font-weight

With the font weight of the text you can choose how bold your font should be. Normally font weight is set in steps of 100. 100 is very thin, 900 very thick. The normal font weight is 400.

NumberProperty

font-size

The font size of the text. This can be set in:

UnitNumberProperty

`cm` `mm` `in` `px` `em` `ex` `ch` `rem`

font-style

The font style of the text. This can be set in:

ListProperty

`normal` `italic` `oblique`

text-transform

You can transform the text with the text transform property. Here, you can set whether the text should automatically be written in upper or lower case and much more. The options are:

ListProperty

`capitalize` `uppercase` `lowercase` `full-width` `none`

font-align

The font align of the text. Here, you can determine whether the text should be aligned left, right or centered. This can be set in:

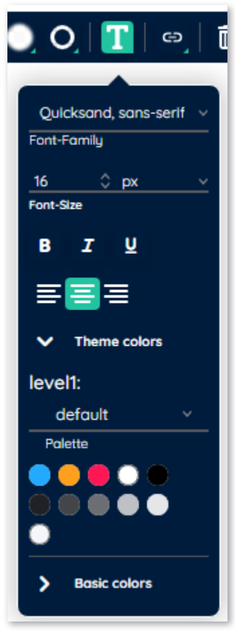
ListProperty

`left` `center` `right`

Occurences

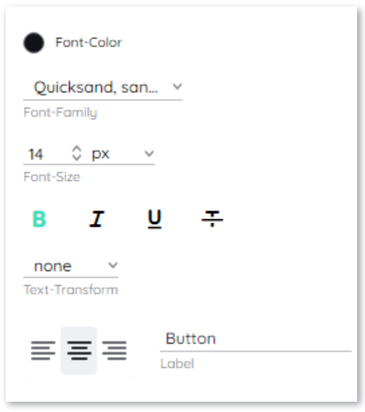
Toolbar

Any availlable font property is grouped. The bar includes Color Picker which allows choosing colors based on the active SubTheme or from basic colors.



Detail Panel

The difference only presents it's self visually in the depiction of the picker. But in detail when opening the Color Picker you can also choose custom colors.



Edit this page

*Last updated by **Felix***

Previous
« Alignment

Next
Element »

Copyright © 2024 YeetDocs, vectorsoft AG.

What is yeet?
Getting Started
UI Components
Essentials
yeet Platform
Advanced
Components Basics
Basic Properties
Multi Picker
Sizing
Spacing
Border
Alignment
Font
Element
OData

Description
Element Ingredients
Occurrences
Detail Panel

Element

Description

The element group can be found Detail Panel. It groups the properties which are used to pick a reference element e.g. choosing an element a yPopup aligns itself with.

Element Ingredients

source-element-parent

This controls the sourceElement the element aligns itself with. If provided and no sourceElementChild is set, it builds the element of alignment. StringProperty

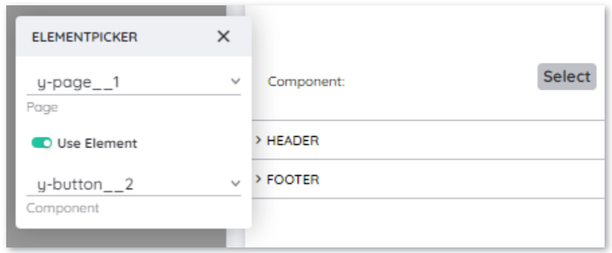
source-element-child

This controls the sourceElement the element aligns itself with. If provided, it outweighs the parent and builds the element of alignment. StringProperty

Occurrences

Detail Panel

Inside the Detail Panel you can select the related element. First you have to choose a parent element, which opens the possibility of choosing one of the children as a reference element.



Edit this page

*Last updated by **Felix***

Previous
« Font

Next
OData »

Copyright © 2024 YeetDocs, vectorsoft AG.