Using the FrosmoPlacement component

After you've installed the Frosmo React Connector to your application project, you can start using the `FrosmoPlacement` component in your application. You wrap every React component whose display rules and content are to be Frosmo-managed inside a `FrosmoPlacement` component. The `FrosmoPlacement` component requires a placement and at least one modification to provide the display rules and content. The association between the `FrosmoPlacement` component and the placement relies on a shared ID, which you must use both as the `id` prop value of the `FrosmoPlacement` component and as the target element selector of the placement. For more information about creating the placement and modifications, see Creating placements and modifications for React.

To use the `FrosmoPlacement` component in a React application:

1. Import `FrosmoPlacement` from the `frosmo-react` module.

   ```
   import { FrosmoPlacement } from 'frosmo-react';
   ```

2. Wrap your Frosmo-managed component inside a `FrosmoPlacement` component. In the following example, `PersonalizedContent` is a custom component, which you would define elsewhere in the React code.

   ```
   <FrosmoPlacement>
       <PersonalizedContent />
   </FrosmoPlacement>
   ```

3. Define an `id` prop for the `FrosmoPlacement` component. The value you set for the prop must also be used for the target element selector of the placement. The `id` prop associates the component to the corresponding placement in the Frosmo Platform.

   ```
   <FrosmoPlacement id="frosmo-placement">
       <PersonalizedContent />
   </FrosmoPlacement>
   ```

4. Optionally, to further specify its behavior, define additional props for the `FrosmoPlacement` component. The following example defines the `DefaultContent` custom component as the default component to render as the content of the `FrosmoPlacement` component.

   ```
   <FrosmoPlacement id="frosmo-placement" defaultComponent={DefaultContent}>
       <PersonalizedContent />
   </FrosmoPlacement>
   ```

   For more information about the available props, see FrosmoPlacement props.
5. Optionally, use the `frosmoModificationContext` prop in a child component to access the content and event tracking functionality of the associated modification. The following example uses the `frosmoModificationContext.content` property to get and display the modification content.

   ```
   class PersonalizedContent extends Component {
       render() {
           const content = {__html: this.props.frosmoModificationContext.content};
           return (
               <div dangerouslySetInnerHTML={content} />
           );
       }
   }

   <FrosmoPlacement id="frosmo-placement" defaultComponent={DefaultContent}>
       <PersonalizedContent />
   </FrosmoPlacement>
   ```

   For more information, see frosmoModificationContext object properties.

FrosmoPlacement props

The following table describes the props that you can define for a `FrosmoPlacement` component.

Table: FrosmoPlacement props

Prop	Description	Type	Role	Example (JSX)
`id`	ID shared by the `FrosmoPlacement` component and the corresponding placement. The React Connector uses the ID to retrieve the correct placement and modification for the component, so the ID you define here must match the target element selector of the placement. The ID can be any string that is a valid prop value, such as "frosmo-placement" or "123456".	String	Required	`id="frosmo-placement"`
`component`	`React.Component` that is rendered if the React Connector successfully retrieves the requested modification. If you define this prop, the content of the `FrosmoPlacement` component is not rendered.	Object	Optional	`component={OverrideContent}`
`defaultComponent`	`React.Component` that is rendered initially, before the React Connector returns the requested modification. If the connector fails to retrieve the modification, for example, because the modification does not exist or the visitor is not in the correct segment, this component remains rendered on the page.	Object	Optional	`defaultComponent={DefaultContent}`

`useFragment`	If you set this prop to `true`, the content of the `FrosmoPlacement` component is rendered as a fragment rather than inside a Frosmo-specific `<div>` element.
	
	The following example shows the same component rendered inside a Frosmo `<div>` element and as a fragment lacking the containing placement `<div>` element:
	
	`<!-- Standard rendering -->`
	
	`<div data-frosmo-elementid="react-placement">`
	`  <div>`
	`  <p>My Frosmo-managed content...</p>`
	`  <p>...straight from a modification!</p>`
	`  </div>`
	`</div>`
	
	`<!-- Fragment rendering -->`
	
	`<div>`
	`  <p>My Frosmo-managed content...</p>`
	`  <p>...straight from a modification!</p>`
	`</div>`
	
	⚠ If you set this prop to `true`, the Frosmo Platform does not automatically track the rendered content for clicks, displays, and true displays. You must explicitly implement the tracking for each child component. Use the tracking functions available in the `frosmoModificationContext` object. For more information, see frosmoModificationContext object properties and Tracking basic events for React fragments.
	
	The default value is `false`.
		Boolean	Optional	`useFragment={true}`

frosmoModificationContext object properties

The following table describes the properties of the `frosmoModificationContext` object, which all children of a `FrosmoPlacement` component receive as a prop.

Table: frosmoModificationContext object properties

Property (data)	Description	Type

`content`	Content of the modification as a single string. Newlines and other control characters in the original content are converted into escape sequences in the returned string.
	
	The following examples show the content of a modification as defined in the modification and as returned in the `content` property:
	
	Original modification content
	
	`<p>My Frosmo-managed content...</p>`
	`<p>...straight from a modification!</p>`
	
	Modification content returned in frosmoModificationContext.content
	
	`"<p>My Frosmo-managed content...</p>\n<p>...straight from a modification!</p>"`
		String

`params`	Template defaults defined for the modification, if any.
	
	The following examples show two template defaults as defined in the modification and as returned in the `params` property.
	
	Template defaults in the modification (JSON object)
	
	`{`
	`  "name": "My Product",`
	`  "price": 99.99`
	`}`
	
	Template defaults in the params property (JavaScript object)
	
	`{`
	`  name: "My Product",`
	`  price: 99.99`
	`}`
	
	The template defaults are also available as props in all children of the `FrosmoPlacement` component. In the following example, `ProductMessage` receives the template defaults `name` and `price` as props from `FrosmoPlacement`.
	
	`<FrosmoPlacement id="frosmo-placement">`
	`  <ProductMessage />`
	`</FrosmoPlacement>`
	
	`class ProductMessage extends Component {`
	`  render() {`
	`  const content = {__html: this.props.name + " costs only " + this.props.price + " euros!"};`
	`  return (`
	`  <div dangerouslySetInnerHTML={content} />`
	`  );`
	`  }`
	`}`
	
	For more information about templates, see Templates.
		Object

`variation`	Number of the modification variation whose content was returned.	Number
Property (function)	**Description**	**Type**

`setTrackableElements`	Function for setting which HTML elements to start tracking for clicks, displays, and true displays when calling `startTracking`. Any tracked events get attributed to the modification. The function takes one parameter: • If you want to track a single element, define an `HTMLElement` representing that element. • If you want to track multiple elements, define an array of `HTMLElements` representing those elements. For more information, see Tracking basic events for React fragments. You only need this function when implementing click, display, and true display tracking for a child component that is rendered as a fragment. You do not need this function if you have no fragments or do not want to track their content for clicks, displays, and true displays.	Function
`startTracking`	Function for starting click, display, and true display tracking for the elements set by `setTrackableElements`. Once started, the Frosmo Platform automatically tracks clicks, displays, and true displays for the specified elements (until the visitor reloads the page, navigates away from the page, or closes the page). Any tracked events get attributed to the modification. For more information, see Tracking basic events for React fragments. You only need this function when implementing click, display, and true display tracking for a child component that is rendered as a fragment. You do not need this function if you have no fragments or do not want to track their content for clicks, displays, and true displays.	Function
`setDisplayed`	Function for manually triggering a display event for the modification. Use this function if a child of the `FrosmoPlacement` component uses a 3rd-party plugin that blocks the normal event tracking of the Frosmo Platform, but you still want to send displays for the child component. The displays get attributed to the modification.	Function
`setTrueDisplayed`	Function for manually triggering a true display event for the modification. Use this function if a child of the `FrosmoPlacement` component uses a 3rd-party plugin that blocks the normal event tracking of the Frosmo Platform, but you still want to send true displays for the child component. The true displays get attributed to the modification.	Function
`setClicked`	Function for manually triggering a click event for the modification. Use this function if a child of the `FrosmoPlacement` component uses a 3rd-party plugin that blocks the normal event tracking of the Frosmo Platform, but you still want to send clicks for the child component. The clicks get attributed to the modification.	Function

Tracking basic events for React fragments

If a `FrosmoPlacement` component is rendered as a fragment, the Frosmo Platform does not automatically track the rendered content for clicks, displays, and true displays. You must explicitly implement the tracking for each child component whose content you want to track.

To track clicks, displays, and true displays for a child component:

1. In the child component, define an `HTMLElement` referencing the content you want to track.
2. Call `frosmoModificationContext.setTrackableElements()` with the `HTMLElement` as the parameter.
3. Call `frosmoModificationContext.startTracking()`.
4. Return the content to be tracked.

The following example shows a simple class component that sets up basic event tracking for the `<div>` element it returns.

```
class PersonalizedContent extends Component {
    componentDidMount() {
        // Get the DOM element reference.
        const el = this.ref.current;
        this.props.frosmoModificationContext.setTrackableElements(el);
        this.props.frosmoModificationContext.startTracking();
    }

    render() {
        return (
            <div ref={ref => { this.ref = ref }}>
                <p>My personalized content.</p>
            );
    }
}

<FrosmoPlacement id="react-placement" useFragment={true}>
    <PersonalizedContent />
</FrosmoPlacement>
```