Widgets

Widgets provide merchants with total control over site content within select areas of the storefront. As a theme developer, you add dropzones to Hypr templates using the `dropzone` tag. Dropzones define areas where merchants can use the Content Editor to drag and drop widgets onto a page (which is only supported for implementations using a Kibo site). Widgets provide any number of useful features, from simple things like an input for raw HTML, to more complex things like adding a carousel view for a list of products. Kibo eCommerce provides many built-in widgets, but you can also add your own custom widgets.

With widgets, merchants can make updates on the fly to keep site content fresh, highlight the latest promotions, make changes to hero images, change header and footer information, upload product videos, and much more.

Widget Components

The basic components that make a widget work are:

Widget Component	Description
Configuration Data	JSON data specified in `theme.json` within the `widgets` array. This data specifies the display name for the widget, the Hypr template the widget uses to render its content, the thumbnail icon for the widget, the page types the widget applies to, and the edit view field or custom editor the widget uses to expose configuration options to Admin users.
Template	The Hypr template the widget uses to render content, located in the `templates/widgets` directory.
Stylesheet	The `.less` file that provides styling rules for the widget, located in the `stylesheets/widgets` directory.
JavaScript (optional)	The script file that provides dynamic functionality to the widget, located in the `scripts/widgets` directory.
Custom Editor (optional)	The Ext.js file that allows you to build more complex configuration menus for your widgets (compared to creating an edit view field in `theme.json`, which satisfies most use cases). Editors are located in the `admin/editors` directory.

Create a Widget

This section walks you through the steps for creating a widget by guiding your through the files that make up an existing widget in the Core theme, the Instock Request Widget. This widget allows shoppers to sign up to be notified when a product is back in stock.

Create a Template for the Widget

Every widget requires a Hypr template to render its content on your site. To learn more about templates, refer to the help topic on Hypr.

1. Navigate to the `templates/widgets` directory in your theme files. This is the folder where you add new widget templates.
2. For purposes of this example, open `instock-request.hypr` in the `misc` directory. The contents of the template look like:

```
{%if SiteContext.IsEditMode %}
Instock Request Widget
{% else %}
   {% if pageModel.productUsage and pageModel.productUsage != 'Bundle' %}
```

What's happening in the template? Essentially, the template renders a form that allows shoppers to subscribe to instock notifications. To accomplish this functionality, this template requires a script file and also includes another template. However, this is just one example, and the exact details of how this template is put together aren't terribly important. Widget templates vary greatly depending on the use case. What's important to remember is that the template is the file where you outline how the widget renders content, and that you have access to the very useful `pageModel` and `model` variables, as detailed in the following table.

`{% if pageModel.productUsage and pageModel.productUsage != 'Bundle' %}`	Uses the `pageModel` global variable available to Hypr templates to determine if a product usage type exists for the page the widget is being viewed on and that the type is not a bundle.
`<span data-mz-instock-request="{% json_attribute model.config %}" style="display:none">` `{% include model.config.template|default:"modules/product/product-instock-request" %}` `</span>`	Looks for a template in the widget configuration data to include and render, and if it doesn't exist, uses the `product-instock-request` template by default. Uses the `model` variable, which unlike `pageModel`, corresponds to the widget instead of the page. More specifically, `model.config` corresponds to the widget configuration data you set in `theme.json` (more on this later).

`{% require_script "widgets/instock-request" %}`	Indicates that this template depends on a specific JavaScript file to achieve its functionality. Note that the `require_script` tag looks for scripts in the `scripts` directory.

3. Open `product-instock-request.hypr.live` in the `templates/modules/product` directory. This is the template included by the widget template. If you examine its contents, you will see that it renders a form that allows a shopper to subscribe to the instock notification (or notifies the shopper that they successfully subscribed).

Set the Widget Configuration Data

The widget configuration data provides the general metadata for the widget and specifies the resources the widget depends on, including the menu options to display to users who drop the widget onto a page.

1. Open `theme.json` in your theme's root directory.
2. Locate the `widgets` array. This array contains a collection of widget configurations.
3. If you are adding a new widget, you would add its configuration data to this array. For this example, locate the widget configuration with a `displayName` equal to `Instock Request`. This is the configuration data for the Instock Request Widget. It looks like:

```
{
  "defaultConfig": {
     "title": "Related Items"
  },
  "displayName": "Instock Request",
  "displayTemplate": "misc/instock-request",
  "editViewFields": [
     {
        "anchor": "100%",
        "fieldLabel": "Title",
        "name": "title",
        "xtype": "mz-input-text"
     }
  ],
  "icon": "/resources/admin/widgets/25_instock_requests.png",
  "id": "instockrequest",
  "validPageTypes": [
     "product"
  ]
}
```

The following table explains the JSON data from the widget configuration.

`defaultConfig`	Contains default values for the widget options, for instances when an Admin user does not provide values of their own. If the default values are used, they are accessible in the widget template through the `model.config` object.
`displayName`	The label name for the widget.
`displayTemplate`	The path to the widget template, starting in the `templates/widgets` directory and omitting the `.hypr` or `.hypr.live` extension.
`editViewFields`	The menu that displays after a user drops the widget onto a page. To learn more about the `xtype` controls that compose `editViewFields`, refer to the reference help. In this example, the widget provides a user-editable field labeled `"Title"`. If the user does not provide a value, then the `defaultConfig` supplies a value of `"Related Items"`. The values a user selects or provides through the widget menu are accessible in the widget template through the `model.config.`*`fieldName`* object. To view every property in the `model.config` object, use the dump tag in the widget template: `{% dump model.config %}`
`customEditor` (not pictured)	If you want more control over the menu that displays to Content Editor users than `editViewFields` can provide, use a custom editor to create a more complex view. For example, `"customEditor": "instockRequest"`, where the `instockRequest` editor would be located in the `admin/editors` directory.
`icon`	The path to the widget icon that displays in the widget selection box.
`id`	The unique identifier for the widget. You can name this whatever you want. The `id` is useful because some widgets have complex JavaScript requirements and you may want to target widget instances within the scripts. As a best practice, you should design your widgets to be tolerant of multiple instances being placed on one page, and the `id` is a handy unique value to refer to the widget.

`validPageTypes`	The types of pages the widget can be added to. You can list multiple types of pages, using the `id` of the page type as identified in the `pageTypes` collection in `theme.json`. For example:
	`[ "home", "product", "404" ]`
	You can also make the widget available to all pages by using an asterisk:
	`[ "*" ]`

You may have noticed that you do not specify a stylesheet or script in the widget configuration data. As you will learn in the following sections, you instead specify a script in the widget template using the `require_script` tag, and you specify a stylesheet in the `storefront.less` file.

Create a Script for the Widget

Although optional, many widgets require JavaScript to accomplish their functionality.

1. Revisit the `instock-request.hypr` template in the `templates/widgets/misc` directory. Note the `{% require_script "widgets/instock-request" %}` tag. This tag looks in the `scripts` directory and marks a script called `instock-request`, omitting the `.js` extension, as required for the widget to function.
2. Open the `instock-request.js` script in the `scripts/widgets` directory. You do not need to understand everything going on in the file, but you can clearly see that you can leverage JavaScript to add advanced functionality to your widget.

Create a Stylesheet for the Widget

When creating a widget, you will likely create new style rules to give your widget the look and feel that you want. The best practice is to create a stylesheet that contains CSS rules unique to your widget, and then import this stylesheet in `storefront.less`.

1. Open `instock-request.less` in the `stylesheets/widgets` directory. This file specifies style rules for the Instock Request widget. It looks like:

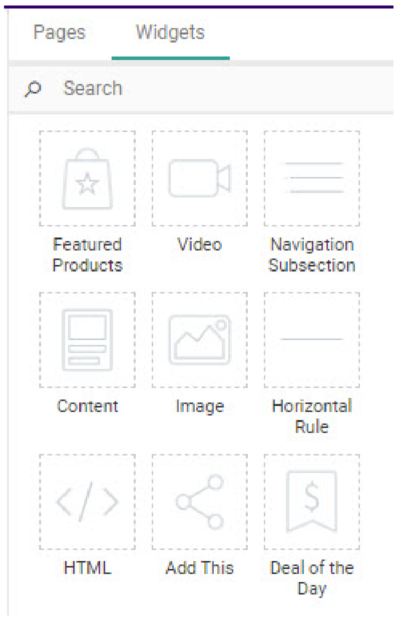
   ```
   .mz-instock-request {
      background: lightgray;
      border: 1px;
      border-style: solid;
      border-color: gray;
      padding: 10px;
      width: 400px;

      button {
         background: limegreen;
         color: white;
         font-weight: bold;

         border:1px;
         border-style: solid;
         border-color: gray;
         border-radius: 5px;
      }
   }
   ```
2. Open `storefront.less` in the `stylesheets` directory. In this file, you specify all the stylesheets that your storefront uses. Notice the section for widgets that imports the Instock Request stylesheet: `@import "/stylesheets/widgets/instock-request.less";`

Test Your Widget

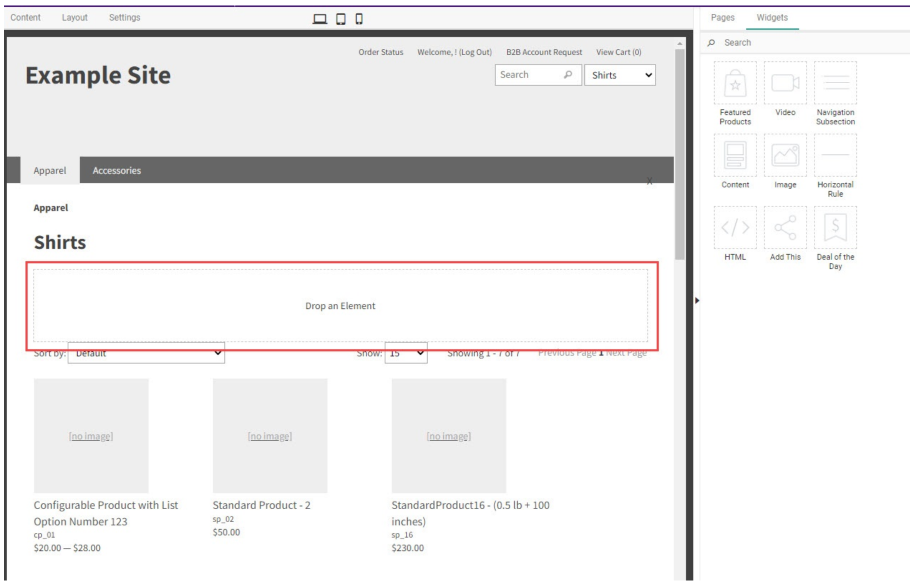
After creating a template, config data, a stylesheet, and a script, you are ready to use your widget!

Normally, after creating a new widget, you would build and upload your theme filesto Dev Center using the Kibo eCommerce theme tools. But for this example, because the Instock Request widget ships with the Core theme, you can gain access to the widget by simply applying the Core theme to your site using the **Themes** page.

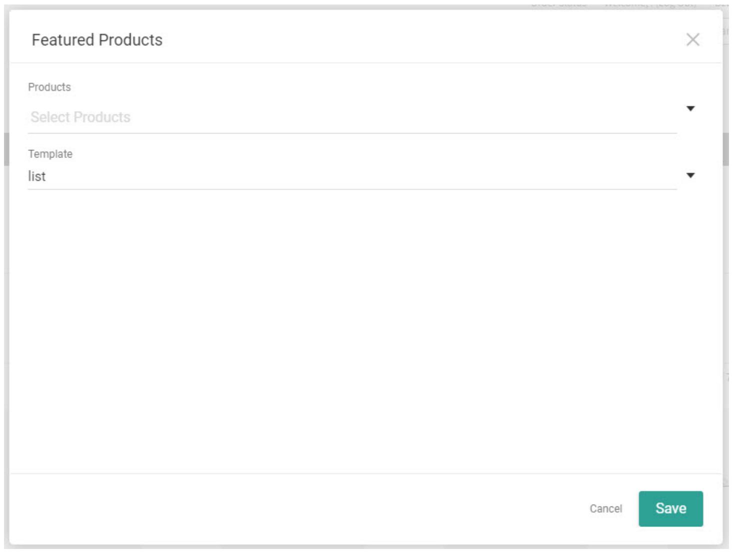
1. Log in to Admin.
2. Go to **Main** > **Content** > **Editor**.
3. Navigate to a product page.
4. Click **Widgets**.



5. Drag and drop the **Instock Request** widget onto a dropzone on the product page.



6. Complete the widget form and click **Save** to close the menu.



7. Click **Save** to save the changes to the product page.
8. Depending on your publishing process, view your new widget on your staged or live site by clicking**View** > **View Live** (or **View Staged**).
9. If the product has zero inventory (which you can set through the**Inventory** page of the Catalog module in Admin), you see the form rendered by the widget that gives shoppers the option to be notified when the product is back in stock:

And that's all there is to creating widgets! Kibo encourages you to try your hand at creating your own custom widgets, which will empower your merchants to create a rewarding shopping experience on the site.