FOR DEVELOPERS

DOCUMOTO WIDGETS IMPLEMENTATION GUIDE

Versions

#	Name	Description	Date
1.0	Alix Bemis	Initial Version for Documoto Implementation Guide for Developers	11-13-24
1.1	Andi Kirtland	Updates to remove Documoto administrative setup options	05-21-25
1.2	Daniel Callif	Updates for Product Pages	06-29-26

Table of Contents

Introduction

This guide provides comprehensive technical instructions for developers implementing Documoto Widgets into an external website or application, such as: dealer portals, eCommerce platforms, or service applications.

The guide focuses on core implementation elements:

- How to securely authenticate and authorize widgets via REST APIs and backend token exchange
- Best practices for setting up and configuring widget properties for different deployment needs
- Customization options including branding, content targeting, and event handling

Prerequisites

Before you begin, ensure the following are in place:

- Widgets are enabled for your Documoto tenant
- Access to a valid Widget API Key
- Determined Widget type (e.g. Library, Media, or Page)
- Relevant content identifiers (as needed)
- Backend server to securely fetch and relay access/refresh token
- Familiarity with JavaScript or your frontend framework
- Basic understanding of REST APIs

Widget Setup

Include the Widget Script

To embed the widget on your website, including the following script tag within your **index.html** or equivalent file:

```
<script src="https://documoto.digabit.com/ui/DocumotoWidget.js">
```

This script retrieves a publicly available JavaScript file from the Documoto website, creating a globally accessible object named **Widget** . This object is required for the widget initialization, authentication, and authorization processes.

Create Widget Container Element

Optionally, you can designate a specific container element in your HTML layout for embedding the Documoto Widget. To do this, assign the **id** attribute the value **documoto-container**:

```
<div  id="documoto-container" style="width: 100%;"/>
```

If this option is not provided in the Widget configuration object, the Documoto Widget will append itself to the HTML body element by default.

Authentication & Token Flow

Widgets use a secure token-based authentication mechanism, requiring token retrieval via a backend service.

Authentication Endpoints

- **Documoto Integration environment**: https://integration.digabit.com/api/ext/authorization/widget/token/v1
- **Documoto Production environment**: https://documoto.digabit.com/api/ext/authorization/widget/token/v1

Generic Access Headers

- **Authorization**: <WIDGET-API-KEY>
- **Content Type**: application/json

Sample Request Body

```
{
    "bindToElementById": "documoto-container",
    "widgetType": "media",
    "mediaIdentifier": "123456789",
    "documotoDomain": "https://documoto.digabit.com",
    "locale": "en-US",
    "enablePartTags": true,
    "enablePartComments": true,
    "theme": {
        "brand_primary": "#d4dfe7"
    }
}
```

Sample Curl Command

```
curl -X POST 'https://documoto.digabit.com/api/ext/authorization/widget/token/v1' \
-H 'Authorization: {WIDGET-API-KEY}' \
-H 'Content-Type: application/json' \
-d '{
    "bindToElementById": "documoto-container",
    "widgetType": "media",
    "mediaIdentifier": "123456789",
    "documotoDomain": "https://documoto.digabit.com",
    "locale": "en-US",
    "enablePartTags": true,
    "enablePartComments": true,
    "theme": {
        "brand_primary": "#d4dfe7"
    }
}'
```

Sample Response

```
{
    "bindToElementById": "documoto-container",
    "widgetType": "media",
    "mediaIdentifier": "123456789",
    "documotoDomain": "https://documoto.digabit.com",
    "locale": "en-US",
    "enablePartTags": true,
    "enablePartComments": true,
    "theme": {
        "brand_primary": "#d4dfe7"
    },
    "accessToken": "**************************************",
    "refreshToken": "**************************************",
    "accessTokenExpiration": "**************************************",
    "refreshTokenExpiration": "**************************************"
}
```

Token by Access Endpoint (Controlled Access)

For tokens bound to specific organizations and user groups

- `POST /api/ext/authorization/widget/token/by-access/v1`

This endpoint requires the following:

- `organizationName` and `userGroupNames` fields in the body of the request
- Setup of Widget User Groups that mirror your existing Documoto Library User Groups
- Controlled Access property enabled in your Documoto tenant (contact your Customer Success Manager)
- Coordination with Documoto Professional Services for setup and enablement guidance

Sample Request Body

```
{
  "bindToElementById":"documoto-container",
  "widgetType": "library",
  "documotoDomain":"<https://documoto.digabit.com  >",
  "locale":"en-US",
  "organizationName": "<desired org name>",
  "userGroupNames": ["<user group name>"]
}
```

Widget Configuration Object

The **config** parameter must be structured as a JavaScript object in the following format:

```
 const config = {
.     widgetType: "library", "media", "page", OR "productPage" // must be a valid
type
      theme: {
        brand_primary: ''<hex value>' // optional
      },
      bindToElementById: "<id of container element>", // optional
      // mediaIdentifier is required for media widget
      // it supports opening to targeted content in library widget
      mediaIdentifier: '<Documoto media identifier of entry point>'
   // productPageIdentifier is required for productPage widget
   productPageIdentifier: '<Documoto product page identifier of entry point>'
      // pageHashkey is required for page widget
      // it supports opening to targeted content in library or media widget
      pageHashKey: '<Documoto page hashKey of entry point>'
      // partNumber is optional; supports opening to targeted content
      partNumber: "<part number of a part in Documoto>"
      documotoDomain: "<customer's Documoto domain e.g.,
https://documoto.digabit.com  >",
      locale: "<locale code> e.g., 'en-US'", // required
      enablePartTags: true,     // optional; defaults to falseif omitted
      enablePartComments: true,  // optional; defaults to false if omitted
    enableInformationLandingPage: true // optional; defaults to false if omitted
      showTableOfContents: true    // optional; defaults to true  if omitted
    }
```

Required Fields by Widget Type

Widget Type	Required Fields
library	• **widgetType**: “library” • **documotoDomain**: the domain for your Documoto tenant (e.g. https://documoto.digabit.com) • **locale**: the locale code for language settings (e.g. en-US)
media	• **widgetType**: “media” • **documotoDomain**: the domain for your Documoto tenant (e.g. https://documoto.digabit.com) • **locale**: the locale code for language settings (e.g. en-US) • **media identifier**: unique id for a media
page	• **widgetType**: “page” • **documotoDomain**: the domain for your Documoto tenant (e.g. https://documoto.digabit.com) • **locale**: the locale code for language settings (e.g. en-US) • **pageHashKey**: unique id for a page
productPage	• **widgetType**: “productPage” • **documotoDomain**: the domain for your Documoto tenant (e.g. https://documoto.digabit.com) • **locale**: the locale code for language settings (e.g. en-US) • productPageIdentifier: unique id for a product page

Initializing the Widget

To initialize the widget, define the widget configuration object on your back-end and include it in the body of a server-side POST request to Documoto’s REST service for authentication.

Upon successful authentication, this REST service will return your configuration object, which will be extended with access and refresh tokens for your session. Your back-end should then pass this extended configuration object to your front-end.

Once your front-end code receives the extended configuration object, you can initialize the widget using the following command within a <script> tag in the HTML of the webpage where the widget will be displayed:

```
Widget.init(config);
```

Wrap this in an async function to maintain security. Avoid exposing your Widget API key or config in the global scope.

Widget Events

Widgets emit events using the CustomEvent API (https://developer.mozilla.org/en-US/docs/Web/API/CustomEvent). The type field identifies the event, while the detail field contains the payload.

Event Types

Event Name	Detail Type	Message	Cause
widgetLogin	Success	Successful Documoto Widget Login	Widget session successfully authenticated and initialized
widgetError	Error	Documoto Widget could not instantiate the iframe	Error occurs during execution of DocumotoWidget.js while setting up the iFrame and initializing Documoto in the front-end
widgetError	Error	Widget could not initialize due to a bad widget configuration	Thrown after Documoto begins initializing within the iFrame if something is missing from the widget config object (e.g., entry point)

widgetError	Error	Failed to Initialize Documoto Widget	Emitted when other initialization failures occur, such as GET requests for user, theme, or tenant properties failing after config is valid
widgetError	Error	Failed to locate the specified content within the widget	Triggered when the widget fails to open to specific target content (instead of the default landing page)

NOTE: Errors do not render an error page but emit events for handling.

Example Detail Payload

```
CustomEvent {
    …
detail: {   "type": "Success",
                "message": "Successful Documoto Widget Login",
                "widgetType": "library"
},
     type: “widgetLogin” OR “widgetError”
}
```

Example Listener

```
  const container = document.getElementById("documoto-container")
  container.addEventListener("widgetLogin", e => {
// Whatever handling code desired goes here
      console.log("widgetLogin event", e)
  })
  container.addEventListener("widgetError", e => {
// Whatever handling code desired goes here
      console.log("widgetError event", e)
  })
```

For a further example, “e” in the first event handler above is structured as follows:

In this case “container” refers to the DOM element that will be parent element of the Documoto Widget. I.e., the one provided in the “bindToElementById” field of your Widget config or document.body by default.

Additionally, “widgetLogin” will catch Event #1 from the aforementioned list in Event Types, and “widgetError” will catch Event #2, #3, #4, and #5.

Add Parts to External Shopping Cart Event

The Documoto Widget can be optionally extended to support adding parts to an external website’s shopping cart. To leverage this feature, be sure to supply the Widget object used to initialize the Widget config with a method called “addToCart”, as

seen below. In this example, “addToExternalCart” is the function/method you would need to add to your codebase or leverage if it already exists elsewhere in your front-end codebase for adding items to the shopping cart.

```
//@params This item is the object from Documoto containing metadata on the item to be
added
function addToExternalCart(item) {
…add your logic here to add item to your cart
}
type item = {
  widgetType: string,
  partId: number,
  partDescription: string,
  pagePartId?: string  //empty when part is added via Search Results Info tab
  partNumber: number,
  quantity: number,
  supplierKey: string,
  uom: string,
  itemText?: string    //empty when part is added via Search Results Info tab
  mediaId?: number     //present when adding part from book context
  mediaIdentifier?: string   //present when adding part from book context
  mediaName?: string   //present when adding part from book context
  chapterId?: number   //present when adding part from book context with chapter
  chapterName?: string //present when adding part from book context with chapter
  pageId?: string//present when adding part from Library page parts BOM
  pageHashkey?: string //present when adding part from Library page parts BOM
  pageName?: string    //present when adding part from Library page parts BOM
}

// Add methods to widget that are used to add to user’s cart
  Widget.addToCart = addToExternalCart
```

Code Implementation Samples

Below are samples of how a Documoto Widget can be implemented within your site. For these implementation approaches, a back-end system is required to handle retrieval of the required access token. This is to prevent storing or exposing the Widget API Key in the website’s front-end code, which can be examined or sniffed and create a security vulnerability or data breach.

Vanilla JavaScript Example

In lieu of React.js, Vue.js, or any other framework approach, the plain JavaScript implementation of a Documoto Widget would look something like the following depending on your pre-existing architecture and context for use. Notably, this example assumes you’d want to initiate the Widget via clicking on a button. However, you can initialize it on page load or any other Document Object Model (DOM) event. In this example, **it is especially critical to ensure the config object and associated calls for it are enclosed within a function to prevent accessing them in the global namespace.** [Closures in JavaScript]

```
<script>
  // Optional -- add event handler logic here to dynamically render the widget
  // Otherwise, just call initWidget directly
  const button = document.getElementById("manual-button")
  button.addEventListener("click", initWidget)

  // Add methods to widget that are used to add to customer's cart
  Widget.addToCart = customerAddToCart

  async function initWidget() {
```

```
    // Optional logic for event handler initialization
    button.removeEventListener("click", initWidget)
    button.remove()
// Get widget config
const config = await getConfig()



    // Call init method on Widget object with config to initialize
    // DocumotoWidget.js brings Widget into scope
    Widget.init(config)

    // Fetch config with tokens for authentication/authorization
    // from back-end server/proxy
    // that uses Widget API key to request tokens from Documoto.
    // Actual path will depend on app configuration.
    // Ensure this function is enclosed so nothing else has access to call it.
    async function getConfig() {
      const res = await fetch("<your API path>")
      const config = await res.json()
      return config
    }
  }
</script>
```

React.js Component Example

The following example code demonstrates how you might support Widgets using React.js. It assumes the desired user experience (UX) is to activate/render the widget by clicking a button. Depending on the level of desired dynamic behavior you might need to pass the appropriate unique identifier to your back-end system for insertion into your widget config object. E.g., when using a Media Widget, you would need to ensure the proper media Identifier is included in the widget config object.

Alternatively, the widget can be simply rendered on mount via removing the button element & showWidget state logic along with adding a useEffect hook to invoke the included config initialization logic on page mount: useEffect – React. Other React.js supported events that can accommodate a handler function for customization could be used as well.

Furthermore, the following implementation patterns require a back-end endpoint to be developed within your system for requesting access tokens via the Documoto REST service at /api/ext/authorization/widget/token/v1. This endpoint would act as a proxy through which your front-end can request the tokens & config object without storing the Widget API Key in JavaScript in the browser.

```
import { useState } from 'react'
// Mocking however app stores back-end url path & constant variables;
// will vary depending on app config, use of axios vs fetch, etc.
import { documotoDomain, tokensPath } from "./const"

// Ensure component & all declared variables are NOT named "Widget"
// to avoid name collision w/Widget global object
function DocumotoWidget() {
  const [ showWidget, setShowWidget ] = useState(false)

  async function handleClick() {
    setShowWidget(state => state = true)

    try {
 // Make the back-end call with the config object to retrieve the hydrated config
object to pass to the widget init.
```

```
Const config = await getConfig()

      // Call init method on Widget object with config to initialize
      // DocumotoWidget.js file imported in index.html brings Widget into scope
      Widget.init(config)
    } catch(e) {
      console.error(e)
    }

    // Fetch tokens for authentication/authorization from back-end server/proxy
    // that uses Widget API key to request tokens from Documoto.
    // Actual path will depend on app configuration.
    // Ensure this function is enclosed so nothing else has access to call it.
    async function getConfig() {
      const res = await fetch(tokensPath)
      const config = await res.json()
      return config
    }
  }

  return (
    // CSS applied to container might need to be tailored based on parent component
CSS
    <section style={{ height: '100vh' }}>
      { !showWidget && (<button onClick={handleClick}>Open manual</button>) }
      <section id="documoto-container" style={{ height: "100%", width: '100%' }}/>
    </section>
  )
}

export default DocumotoWidget
```

Note: This implementation and the Vue.js example below both still require the DocumotoWidget.js script to be included in the site's index.html for the global Widget variable to be present:
<script src="https://documoto.digabit.com/ui/DocumotoWidget.js"></script>

Vue.js Component Example

If your website's front-end is developed with Vue.js, then the equivalent code from the React.js example would look like the following component. As with React.js, this example can be flexibly tailored to meet your needs via handling different supported Document Object Model (DOM) events with handler functions. Additionally, the widget can just simply be rendered on page mount via the Vue.js onMounted lifecycle hook: Vue.js onMounted.

```
<template>
  <section class="outer-container">
    <button   v-if="!showWidget"
              @click="handleClick">Open manual</button>
    <section  id="documoto-container"
              class="inner-container"/>
  </section>
</template>

<script setup>
// Ensure component & all declared variables are NOT named "Widget"
// to avoid name collision w/Widget global object
import { ref } from "vue"
import { documotoDomain, tokensPath } from "../const"
```

```
let showWidget = ref(false)

async function handleClick() {
  showWidget.value = true
  try {
const config = await getConfig()
    // Call init method on Widget object with config to initialize
    // DocumotoWidget.js file imported in index.html brings Widget into scope
    Widget.init(config)
  } catch(e) {
    console.error(e)
  }

  // Fetch tokens for authentication/authorization from back-end server/proxy
  // that uses Widget API key to request tokens from Documoto.
  // Actual path will depend on app configuration.
  // Ensure this function is enclosed so nothing else has access to call it.
  async function getConfig() {
    const res = await fetch(tokensPath)
    const config = await res.json()
    return config
  }
}

</script>

<style scoped>
.outer-container {
  height: 100vh;
}
.inner-container {
  height: 100%;
  width: 100%;
}
</style>
```

Important Considerations

Recommended Sizes for Widgets

As a best practice, the Widget should be allotted the maximum width of a web page wherever possible. This allows for an optimal mobile browsing experience as well.

Limitations

To ensure content security within Documoto and Documoto Services, Widgets do not currently support exporting or downloading content of any kind. To support exporting and downloading content in your site along side of a Widget, it is recommended to leverage Documoto's library of REST APIs, which can be referenced in the publicly available Swagger Specification.

For assistance with setting up and leveraging Documoto's REST APIs, see the following Knowledge Base articles or contact your Customer Success Manager.

- REST Web Services Reference
- How to Export Content via REST Web Services

Note: Depending on your Documoto subscription agreement, throttling rate and API usage varies. Contact your Customer Success Manager for more information on your subscription's REST API Rate & Usage limits.

Terms & Conditions of Use for Documoto Widgets

By using a Documoto Widget, you explicitly agree to assume all responsibility for implementing the widget within your website including, but not limited to, implementing the necessary code to facilitate the widget, and implementing adequate security measures on your website. You acknowledge that Documoto shall not be held liable for any security breaches of your website.

You acknowledge and agree that as the end user, it is your responsibility to implement appropriate security measures on your website to protect against unauthorized access, use, or disclosure of the Documoto content through the Widget. Documoto shall not be liable for any security breaches or data breaches resulting from the improper implementation of security measures by you or any third party acting on your behalf. You expressly acknowledge that the security and protection of data transmitted through the Widget depend on the measures you implement.

Documoto will make every effort available to maintain platform stability and performance, including scheduled maintenance, improvements, and fixes as deemed necessary by Documoto. You acknowledge and agree that the performance of the widget, as it pertains to your website, is dependent on the implementation approach and the overall performance of the containing website.