Widget integration guide

DPD Checkout & Return widgets

Widget integration guide

DPD Checkout & Return widgets

Table of contents

Widget integration guide

DPD Checkout & Return widgets

1. Brief description

This document contains the implementation guide for the checkout & return widgets of DPD.
It is specifically designed for webshop maintainers to help them integrate the checkout and return widgets in their webshop.

2. Version history

The following table shows the version history of this document.

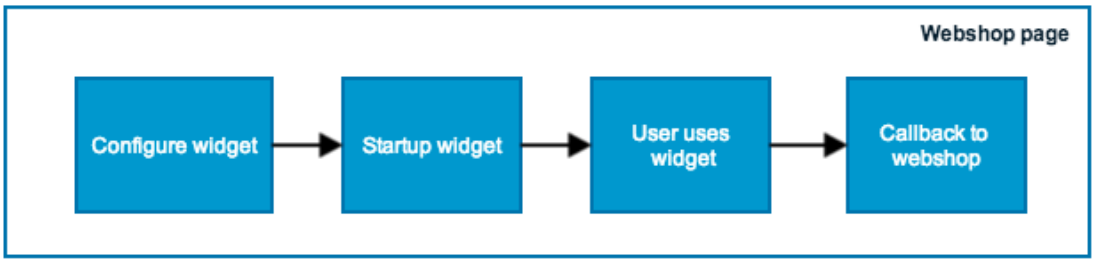
Version	Date	Changes
1.0	May 9th, 2014	Initial version
1.1	September 8th, 2014	Added section 4.4 on order confirmation

3. Common

This chapter describes aspects that are common to both the checkout and the return widget, like flow of interaction between shop and widget, authentication, data validation and format of data exchanged between widget and webshop.

3.1. Widget flow

The webshop initializes the widget by including the widget JavaScript file. The widget then needs to be initialized with the widget configuration.
After the widget has been initialized, it can be used by the customer. When the customer interacts with the widget a callback to the webshop is made to signal choices from the user back to the webshop.



Each widget can run in 2 modes (Inline and Overlay). When the Inline option is used, the widget is embedded in the webshop page. A widget started in overlay will float on top of the webpage and can be closed with a close button.

3.2. Authorization & Validation

In each widget an api key is used to authorize the webshop for using the widget. This api key is a UUID that is passed in the configuration of the widget. The api key is distributed by DPD.
At startup the current domain-name of the webshop is validated with a list of allowed domains to guarantee that the widget can't be started on non-permitted websites.
The widget configuration is also checked for required values. The widget will show an error on startup when the domain name isn't valid or required values are missing.

3.3. Configuration format

Each widget uses the JavaScript Object Notation (JSON) as input / output for the widgets. See http://json.org/ for the JSON specification.
The following types are used in this document:

Type	Description	Example
object	A JSON object (with properties)	{ “key”: “value” }
array	An array containing zero or more values.	[1, 2, 3]
string	A string value	“value”
int	An integer value	123
double	A decimal value	32.5
boolean	A Boolean value	true or false

All keys in JSON objects are written in camel case, with the first word starting with a lowercase letter.

4. Checkout widget

This chapter describes the configuration and usage of the checkout widget.

4.1. Configuration format

The checkout widget needs to be called with a number of configuration options on startup.
The following table contains the options that can be configured.

Name	Description	Type	Constraints
integrationType	The way the widget is integrated	string	Value must be “Inline” or “Overlay”
locale	The locale of the widget (texts)	string	Optional
apiKey	The api key of the webshop	string	UUID format
divId	The id of the div where the widget gets loaded	string	Not null
order	The webshop order	object	Not null
- orderNumber	The order number of the order	string	Not null
- orderWeight	The weight of the order in KG’s	double	Not null, >= 0
- orderPrice	The price of the order in euros.	double	Not null, >= 0
consumerInfo	Info about the person placing the order	object	Not null
- name	Name of the consumer	string	
- email	Email of the consumer	string	Not null, valid email-address
- phoneNumber	Phone number of the consumer	string	Valid phone number or null
- address	Address info of the consumer	object	
-- street	Street name of the consumer	string	
-- houseNumber	House number of the consumer	string	
-- houseNumberAddition	House number addition of the consumer	string	
-- postalCode	Postal code of the consumer	string	
-- city	City of the consumer	string	
-- countryISO	Country of the consumer	string	

At startup a mandatory callback function has to be configured as well. This function is called when the widget returns information to the webshop.

The callback function has the following signature:

```
function (data) { }
```

The callback function is called each time the user enters data that changes the state of the widget. The data format is described in the widget callback chapter.

4.2. Startup

Include a div on the page on the location of where the widget needs to be displayed:

```
<div id="divId"></div>
```

Include the following script in the bottom of the body of your webpage:

```
<script src="https://widgets.dpd.nl/widgets/Scripts/checkoutWidget/init.min.js”></script>
<script>
	var config = {}; // Your widget config
	function callback (data) {
		// Callback implementation
	}
	startCheckoutWidget(config, callback);
</script>
```

The JavaScript code for configuring and starting the widget can also be placed in a function that is called on the onload event of the body.

Please note that the checkout widget uses 360px by 450px in Inline mode.

4.3. Widget callback

When the widget finishes it emits a callback to the webshop. The format of the callback data is described below:

Name	Description	Type	Constraints
product	The chosen product	string	Value is “DPDHome” or “DPD-ParcelShop”
consumerInfo	Info about the person placing the order	object	Not null
- name	Name of the consumer	string	Not null
- email	Email of the consumer	string	Not null, valid email-address
- phoneNumber	Phone number of the consumer	string	Valid phone number or null
- address	Address info of the consumer	object	Not null
-- street	Street name of the consumer	string	Not null

Name	Description	Type	Constraints
-- houseNumber	House number of the consumer	string	Not null
-- houseNumberAddition	House number addition of the consumer	string	
-- postalCode	Postal code of the consumer	string	Not null, For NL a valid postal code
-- city	City of the consumer	string	Not null
-- countryISO	Country of the consumer	string	Not null
parcelShop	The ParcelShop	object	Not null
- id	The id of the ParcelShop	int	Not null
- name	The name of the ParcelShop	string	
- address	Address info of the ParcelShop	object	Not null
-- street	Street name of the ParcelShop	string	Not null
-- houseNumber	House number of the Par-celShop	string	Not null
-- houseNumberAddition	House number addition of the ParcelShop	string	
-- postalCode	Postal code of the ParcelShop	string	Not null
-- city	City of the ParcelShop	string	Not null
-- countryISO	Country of the ParcelShop	string	Not null
- openingHours	List of opening hours	array	
-- [item]	Opening hours object	object	
--- weekday	The day of the week	string	Day in English
--- openMorning	Open time	string	HH:MM
--- closeMorning	Close time	string	HH:MM
--- openAfternoon	Open time	string	HH:MM
--- closeAfternoon	Close time	string	HH:MM
- services	Supported services	object	Not null
-- pickupByConsigneeAl-lowed	Is pickup allowed?	Boolean	Not null
-- returnAllowed	Are returns allowed?	boolean	Not null
-- expressAllowed	Is express shipping allowed?	boolean	Not null
-- prepaidAllowed	Is prepaid payment allowed?	boolean	Not null
-- cashOnDeliveryAllowed	Is cash on delivery allowed?	Boolean	Not null
- longitude	Longitude of the ParcelShop	double	
- latitude	Latitude of the ParcelShop	double	
order	The webshop order	object	Not null
- orderNumber	The ordernumber of the order	string	
- orderWeight	The weight of the order in KG's	double	Not null, >= 0
- orderPrice	The price of the order in euros	double	Not null, >= 0
- shipping	The shipping data	object	Not null
-- price	The shipping price	double	Not null
-- verifier	The verifier	string	Not null if a shared secret has been configured

The shipping price returned by the callback can be verified by the verifier field.
You can calculate the verifier by using the following formula:

Base64(HMAC-SHA256(shippingPrice + productType + country + orderPrice + orderWeight, secret))

Example:
shippingPrice = 3
productType = DPDHome
country = NL
orderPrice = 12.95
orderWeight = 2.5
secret = passw0rd

c = 3.00DPDHomeNL12.952.50
verifier = Base64(HMAC-SHA256(c, secret)) = hkoBxt5cHAlkR/JRY1mKSqpj3O39QRHn+smibajgdzk=

Please note the following:

- All numeric values are rounded to (exactly) 2 decimal places. So for example the number “3” becomes “3.00”;
- The country is the country for the selected product. For DPDParcelShop this is the ParcelShop country, for DPDHome this is the country from the consumer info.

Example implementations can be found in the examples chapter.

4.4. Order confirmation

When a user selects a shipping option in the checkout widget, the details of the order are passed to the maintenance environment. This is being done to make it easier for the webshop maintainer to generate labels.

At the time the shipping option gets selected it is unknown whether the order is actually going to be completed by the customer. Therefore a separate confirmation callback can be used on the order confirmation page of the webshop.

The following script can be used to confirm the order:

```
<script src="https://widgets.dpd.nl/widgets/Scripts/checkoutWidget/init.min.js"></script>
<script>
	var config = {apiKey: 'API_KEY', orderNumber: 'ORDER_NUMBER'};
	confirmOrder(config);
</script>
```

The preferred place for the script is at the end of the page body.

5. Return widget

This chapter describes the configuration and use of the return widget.

5.1. Widget startup

The return widget needs to be called with a number of configuration options on startup.
The following table contains the configuration options that can be configured.

Name	Description	Type	Constraints
integrationType	The way the widget is integrated	string	Value must be “Inline” or “Overlay”
locale	The locale of the widget (texts)	string	Optional
apiKey	The api key of the webshop	string	UUID format
divId	The id of the div where the widget gets loaded	string	Not null
referenceNumber	The return reference, as issued by the webshop. Can for example be an RMA number.	string	Not null
consumerInfo	Info about the person placing the order	object	Not null
- name	Name of the consumer	string	
- email	Email of the consumer	string	Not null, valid email-address
- phoneNumber	Phone number of the consumer	string	
- address	Address info of the consumer	object	
-- street	Street name of the consumer	string	
-- houseNumber	House number of the consumer	string	
-- postalCode	Postal code of the consumer	string	
-- city	City of the consumer	string	
-- countryISO	Country of the consumer	string	

At startup a mandatory callback function has to be configured as well. This function is called when the widget returns information to the webshop.
The callback function has the following signature:

```
function (data) { }
```

The callback function is called each time the user enters data that changes the state of the widget. The data format is described in the widget callback chapter.

5.2. Startup

Include a div on the page on the location of where the widget needs to be displayed:

```
<div id="divId"></div>
```

Include the following script in the bottom of the body of your webpage:

```
<script src="https://widgets.dpd.nl/widgets/Scripts/returnWidget/init.min.js"></script>
<script>
	var config = {}; // Your widget config
	function callback (data) {
		// Callback implementation
	}
	startReturnWidget(config, callback);
</script>
```

The JavaScript code for configuring and starting the widget can also be placed in a function that is called on the onload event of the body.

Please note that the return widget uses 520px by 620px in Inline mode.

5.3. Widget callback

When the widget finishes it emits a callback to the webshop. The format of the callback data is described below:

Name	Description	Type	Constraints
referenceNumber	The return reference	string	Not null
shipmentNumber	The parcel label number	string	Not null
consumerInfo	Info about the person placing the order	object	Not null
- name	Name of the consumer	string	Not null
- email	Email of the consumer	string	Not null, valid email-address
- phoneNumber	Phone number of the consumer	string	Valid phone number or null
- address	Address info of the consumer	object	Not null
-- street	Street name of the consumer	string	Not null
-- houseNumber	House number of the consumer	string	Not null
-- houseNumberAddition	House number addition of the consumer	string	
-- postalCode	Postal code of the consumer	string	Not null, For NL a valid postal code
-- city	City of the consumer	string	Not null
-- countryISO	Country of the consumer	string	Not null

Name	Description	Type	Constraints
parcelShop	The ParcelShop	object	Not null
- id	The id of the ParcelShop	int	Not null
- name	The name of the ParcelShop	string	
- address	Address info of the ParcelShop	object	Not null
-- street	Street name of the ParcelShop	string	Not null
-- houseNumber	House number of the ParcelShop	string	Not null
-- houseNumberAddition	House number addition of the ParcelShop	string	
-- postalCode	Postal code of the ParcelShop	string	Not null
-- city	City of the ParcelShop	string	Not null
-- countryISO	Country of the ParcelShop	string	Not null
- openingHours	List of opening hours	array	
-- [item]	Opening hours object	object	
--- weekday	The day of the week	string	Day in English
--- openMorning	Open time	string	HH:MM
--- closeMorning	Close time	string	HH:MM
--- openAfternoon	Open time	string	HH:MM
--- closeAfternoon	Close time	string	HH:MM
- services	Supported services	object	Not null
-- pickupByConsigneeAllowed	Is pickup allowed?	boolean	Not null
-- returnAllowed	Are returns allowed?	boolean	Not null
-- expressAllowed	Is express shipping allowed?	boolean	Not null
-- prepaidAllowed	Is prepaid payment allowed?	boolean	Not null
-- cashOnDeliveryAllowed	Is cash on delivery allowed?	boolean	Not null
- longitude	Longitude of the ParcelShop	double	
- latitude	Latitude of the ParcelShop	double	

6. Examples

6.1. Checkout widget configuration

```
{
   "integrationType": "Inline",
   "locale": "nl_NL",
   "apiKey": "ec4761a4-885e-475f-b9ac-8309dc08e651",
   “divId”, “someDiv”,
   "order": {
      "orderNumber": 123,
      "orderPrice": 15.0,
      "orderWeight": 2.5
   },
   "consumerInfo": {
      "name": "Harry de Vries",
      "email": "test@test.nl",
      "phoneNumber": null,
      "address": {
         "street": "Lage biezenweg",
         "houseNumber": "1",
         "houseNumberAddition": null,
         "postalCode": "4131LV",
         "city": "Vianen",
         "countryISO": "NL"
      }
   }
}
```

6.2. Return widget configuration

```
{
   "integrationType": "Inline",
   "locale": "nl_NL",
   "apiKey": "ec4761a4-885e-475f-b9ac-8309dc08e651",
   “divId”, “someDiv”,
   "referenceNumber": "abcd123",
   "consumerInfo": {
      "name": "Harry de Vries",
      "email": "test@test.nl",
      "phoneNumber": null,
      "address": {
         "street": "Lage biezenweg",
         "houseNumber": "1",
         "houseNumberAddition": null,
         "postalCode": "4131LV",
         "city": "Vianen",
         "countryISO": "NL"
      }
   }
}
```

6.3. Calculate verifier

C#

```
byte[] key = Encoding.UTF8.GetBytes("passw0rd");
decimal shippingPrice = 3;
decimal orderAmount = 12.95;
double orderWeight = 2.5;
string country = "NL";
string productType = "DPDHome";

using (HMAC hmac = new HMACSHA256(key))
{
    StringBuilder data = new StringBuilder()
        .Append(shippingPrice.ToString("F2", CultureInfo.InvariantCulture))
        .Append(productType)
        .Append(country)
        .Append(orderAmount.ToString("F2", CultureInfo.InvariantCulture))
        .Append(orderWeight.ToString("F2", CultureInfo.InvariantCulture));

    byte[] hash =
hmac.ComputeHash(Encoding.UTF8.GetBytes(data.ToString()));
    string verifier = Convert.ToBase64String(hash);
```

Java

```
double shippingPrice = 3;
double orderAmount = 12.95;
double orderWeight = 2.5;
String country = "NL";
String productType = "DPDHome";
DecimalFormat format = new DecimalFormat("0.00");
StringBuilder data = new  StringBuilder()
            .append(format.format(shippingPrice))
            .append(productType)
            .append(country)
            .append(format.format(orderAmount))
            .append(format.format(orderWeight));
String verifier = hmacDigest(data.toString(), "passw0rd", "HmacSHA256");

// hmacDigest method
public static String hmacDigest(String msg, String keyString, String algo) {
    try {
        SecretKeySpec key = new SecretKeySpec((keyString).getBytes("UTF-8"),
algo);
        Mac mac = Mac.getInstance(algo);
        mac.init(key);
        byte[] bytes = mac.doFinal(msg.getBytes("UTF-8"));
        return Base64.encodeBase64String(bytes);
    } catch (Exception e) {
        // Help
    }
    return null;
}
```

PHP

```
$secret = "passw0rd";

$shippingPrice = 3;
$orderAmount = 12.95;
$orderWeight = 2.5;
$country = "NL";
$productType = "DPDHome";

$data = number_format($shippingPrice, 2) . $productType . $country .
        number_format($orderAmount, 2) . number_format($orderWeight, 2);
$verifier = base64_encode(hash_hmac('sha256', $data, $secret, true));
```