Label Templates

Label templates for COBI.wms are HTML files which may contain special placeholder values interpreted by COBI.wms. The placeholders will be replaced with their actual values in a simple search-and-replace manner before the HTML is passed to the Android system for graphical rendering.

Accepted placeholder values

The following sections explain the different kinds of placeholders that are accepted in COBI.wms label templates.

Simple placeholders

The following table describes all simple placeholders understood by the app. Each of these must appear within the HTML document in the format `@placeholderGoesHere@`, i.e. with an @-symbol before and after the name of the placeholder, with no space characters in between.

Document-level	
`docNumber`	Document number
`docForeignNumber`	BP-reference of document
`docDate`	Posting date
`docDueDate`	Due date
`docReference`	The “Reference 2” field that some documents have
`docComments`	The comments field
Based on document's business partner	
`businessPartnerCode`	Code of business partner
`businessPartnerName`	Name of business partner
Line-level	
`lineNumber`	The line number **starting from zero**
`lineItemName`	Description (normally contains item name)
`lineFreeText`	Free text
`lineUnitName`	UoM name as entered in document line
`lineBPCatalogCode`	Business partner catalog number (`SubCatNum`)
`lineBPCatalogName`	Business partner catalog description (`OSCN.Descriptio`)
Based on line's unit	
`unitCode`	UoM code (if not “Manual”)
`unitName`	UoM name (ignores manually entered name in document line)
`unitQtyInBaseUnit`	Quantity of this unit measured in base unit
Item-level	
`itemCode`	Item code
`itemName`	Item name
`itemForeignName`	Foreign name of item (often used for an alt. item code)
`itemBarcode`	Primary barcode of the item, as text

Document-level	
itemBarcodeGTIN12	Primary barcode, fitted into GTIN-12 format
itemBarcodeGTIN13	Primary barcode, fitted into GTIN-13 format
itemBarcodeGTIN14	Primary barcode, fitted into GTIN-14 format
itemSupplierCatalogNumber	Item number in supplier's catalog
itemAdditionalIdentifier	Additional identifier (database field SWW)
itemGroupName	Name of the item group in which the item is
itemPurchasePackQty	Quantity of items in one purchase package
itemSalesPackQty	Quantity of items in one sales package
itemPurchaseUnitQty	Stock quantity per purchased unit
itemSalesUnitQty	Stock quantity per sold unit
itemPurchaseUnitName	Name of the purchase UoM
itemSalesUnitName	Name of the sales UoM
itemStockUnitName	Name of the stock UoM
itemPurchaseLength	Length of the purchase UoM
itemPurchaseWidth	Width of the purchase UoM
itemPurchaseHeight	Height of the purchase UoM
itemPurchaseLengthUnit	Unit in which the purchase UoM length is given
itemPurchaseWidthUnit	Unit in which the purchase UoM width is given
itemPurchaseHeightUnit	Unit in which the purchase UoM height is given
itemSalesLength	Length of the sales UoM
itemSalesWidth	Width of the sales UoM
itemSalesHeight	Height of the sales UoM
itemSalesLengthUnit	Unit in which the sales UoM length is given
itemSalesWidthUnit	Unit in which the sales UoM width is given
itemSalesHeightUnit	Unit in which the sales UoM height is given
itemPurchaseUnitContents	Contents of the purchase UoM in terms of the base UoM of the UoM group
itemSalesUnitContents	Contents of the sales UoM in terms of the base UoM of the UoM group
itemStockUnitContents	Contents of the stock UoM in terms of the base UoM of the UoM group
Batch-level	
batchNumber	Batch number
batchProductionDate	Production date of the batch
batchProductionGS1	Production date in GS1 barcode format (YYMMDD)
batchExpiryDate	Expiry date of the batch
batchExpiryGS1	Expiry date in GS1 barcode format (YYMMDD)
batchAttr1	Batch attribute 1
batchAttr2	Batch attribute 2
batchDetails	Batch details/notes
Serial-level	
serialNumber	Serial number
serialMnfNumber	Manufacturer serial number
serialLotNumber	Lot-number of serial number

Document-level	
`serialDetails`	Serial details/notes
Bin location-level	
`locationCode`	Bin location code
`locationBarcode`	Bin location barcode
Context-dependent	
`quantity`	Quantity of selected line, batch, etc.

The placeholders on document- and line-level are only valid when the label printing is triggered on a document line, such as while creating or after booking a document.

The placeholders of the form `itemBarcodeGTIN…` will try to force the default barcode of the item into the specified number of digits by either appending or removing zeroes at the left, such that the actual value of the GTIN number doesn't change. If this conversion is impossible, for instance if you try to use `itemBarcodeGTIN13` but the barcode uses a full 14 digits with no leading zeroes, then a pseudo-GTIN of all-zeroes will be used. The utility of these special placeholders becomes apparent in the next section.

Item prices

You can use placeholders of the form `@itemPrice(x)@` to display the price of an item in the price list number `x`.

The resulting text will contain the price value with two decimal positions as well as the three-letter currency code, for example: `5.00 USD`

The decimal separator that is used depends on the language settings of the Android device, so for example a device set to German would format the same as `5,00 USD`.

Additionally, the following variants are supported for fine-tuning the format:

Placeholder	**Description**
`itemPriceRawValue`	Price value formatted via Java's `Double.toString()` method, e.g. `5.5`.
`itemPriceFormattedValue`	Price value formatted according to device language, e.g. `5.50` or `5,50`.
`itemPriceCurrencyCode`	The three-letter currency code of the price's currency, e.g. USD or EUR.
`itemPriceCurrencySymbol`	A symbol representing the price's currency, e.g. $ or €.

Examples, where the price is 5 US dollars and 50 cents, and the language is set to US English:

- `@itemPrice(1)@` → `5.00 USD`
- `@itemPriceFormattedValue(1)@ @itemPriceCurrencyCode(1)@` → `5.00 USD` (same as above)
- `@itemPriceCurrencySymbol(1)@@itemPriceValue(1)@` → `$5.00`

The placeholder `itemPriceRawValue` is especially useful if the price is going to be used in JavaScript for calculations, where it's important to get its value in a consistent format so it can be parsed as a number.

Barcode graphics

The following special type of placeholder is supported to print actual barcodes on the label:

```
@barcode(FORMAT,WIDTH,HEIGHT,CONTENT)@
```

NOTE: Don't put any spaces before or after the surrounding parentheses or the commas separating the fields!

The following values are supported in place of the `FORMAT` parameter:

- CODEBAR
- CODE_39
- CODE_93
- CODE_128
- DATA_MATRIX
- EAN_8
- EAN_13
- QR_CODE
- UPC_A
- UPC_E

The values given for the `WIDTH` and `HEIGHT` parameters are merely guidelines for the barcode generator. If the generated barcode doesn't fit in the given width/height, they will be exceeded.

For the `CONTENT` parameter, you can use any other placeholder, free text, or a combination thereof.

Here's a full example:

```
@barcode(EAN_13,100,50,@itemBarcodeGTIN13@)@
```

This placeholder will be replaced with an EAN-13 barcode, containing the primary barcode of the item according to the barcodes table in the item master data in SAP Business One.

Since we've used the placeholder `itemBarcodeGTIN13`, the barcode of the item may actually be a GTIN-12, or it may be saved as a GTIN-14 in SAP Business One with a leading zero. In both cases, the EAN-13 barcode will be generated just fine, since the `itemBarcodeGTIN13` adds or removes zeroes as necessary. If the conversion to the correct number of digits is not possible (e.g. barcode is a full-length GTIN-14, so there's no leading zero to remove), then the value `0000000000000` (13 zeroes) will be used instead; this way it's ensured that an EAN-13 barcode is still generated on the label without messing up the whole print layout. The printed barcode will of course be non-functional.

GS1 Support

The formats `CODE_128`, `DATA_MATRIX`, and `QR_CODE` support generating GS1 Barcodes.

To make use of this, the contents of the barcode must begin with the vertical bar symbol: **|**

GS1 fields with variable length that need to be terminated are likewise terminated with a vertical bar symbol.

Example:

```
@barcode(CODE_128,100,75,|01@itemBarcodeGTIN14@10@batchNumber@|...)@
```

In this example, the first GS1 barcode field is `01` which symbolizes that a GTIN-14 follows. By using the placeholder `itemBarcodeGTIN14` we ensure that exactly 14 digits are inserted here. This field does not need to be terminated with a vertical bar symbol, since it has a fixed length of 14 digits.

The next field is `10` i.e. the batch number. This field is allowed to contain 1 to 20 digits or arbitrary characters. If we use batch numbers of 10 characters, this means we must end this GS1 field explicitly with a vertical bar symbol.

Here's a list of commonly used GS1 field identifiers:

https://www.activebarcode.de/codes/ean128_ucc128_ai.html

Here's a full list from GS1.org currently containing 480 entries:

https://www.gs1.org/standards/barcodes/application-identifiers

Better control of generated HTML

The `barcode()` placeholder generates a whole `<img … />` tag in the generated HTML. If you want closer control over the HTML, like adding additional attributes to the `img` tag, you can use the `barcodeSrc()` or `barcodeBase64()` placeholders instead.

The `barcodeSrc` placeholder generates only the contents of the `src` attribute. Example:

```
<img id='...' class='...' src='@barcodeSrc(...)@' />
```

The `barcodeBase64` placeholder generates only the Base64 string. Example:

```
<img id='...' class='...' src='@barcodeBase64(...)@'
/>
```

The parameters of the `barcodeSrc()` and `barcodeBase64()` placeholders are exactly the same as that of the regular `barcode()` placeholder. That means you must still provide the `WIDTH` and `HEIGHT` parameters; these will be passed to the barcode generation system that produces the Base64 PNG. (This has no effect on the width and height of the `img` element in HTML/CSS.)

User-input values

Label templates may contain a special type of placeholder that tells the app to show the user a popup for values to insert manually every time the label is to be printed. The simplest syntax for this type of placeholder looks as follows:

```
@input(NAME)@
```

The `NAME` parameter uniquely identifies the input and may only consist of normal English letters and

digits. (No spaces, special characters, umlauts, etc.)

The following logic is applied to label templates with input placeholders:

- The template is scanned for all input placeholders.
- Whenever such a placeholder is found and the NAME has not been encountered before, it's registered into a list of needed inputs.
- The user is then presented with a popup that contains one field per needed input. The value provided by the user is saved.
- Finally, all input placeholders are replaced with the value that the user had entered for them. Placeholders with the same NAME get the same value.

Since the name of an input may not contain arbitrary text, an optional “display name” may be specified in square brackets, which will be used in the popup where the user has to enter the value for that input. Example:

```
@input(quantity[Number of crates])@
```

In this example, the input is identified by the simple name `qty` but the COBI.wms user will see it as “Number of crates” in the input values popup. If no display name is specified in square brackets, then the regular name of the input is also used as the display name.

Also, an input may be limited to a predefined set of choices by listing them in the following format:

```
@input(Currency;EUR,GBP,USD)@
```

NOTE: Don't put any spaces after the semicolon, or before or after the commas.

This means that in the input popup shown to the user, there will be a drop-down list of selections instead of a free-form text field.

The display name and the list of choices can be combined:

```
@input(shipType[Freight type];Less Than Truckload,Full Truckload,Air
Freight,Ocean Freight)@
```

The configuration for an input (display name and list of values) only has to appear for the first time the input with that NAME is encountered. A good strategy is to list all the inputs with their configuration at the top of the HTML file within an HTML comment, and only refer to them as `@input(NAME)@` in the main body of the HTML, to make the template easier to read. Refer to the second example template for an example of this strategy.

Date placeholders

You can insert the date and/or time of the moment of the printing by using the following special placeholder:

```
@date(FORMAT)@
```

Valid values for the FORMAT parameters correspond to the parameters of the Android class

`SimpleDateFormat` which is documented in the following page:

https://developer.android.com/reference/java/text/SimpleDateFormat

However, for your convenience, here are the values that are likely to be used most commonly:

yyyy	Year, 4 digits
yy	Year, 2 digits
MM	Month number, 2 digits
dd	Day of month, 2 digits
HH	Hour of day, 2 digits
mm	Minute of hour, 2 digits
ss	Second of minute, 2 digits

Examples follow. Let's say the date is February 23rd, 2021, and the clock reads 16:11.

`@date(yyyy-MM-dd HH:mm)@` becomes `2021-02-23 16:11`

`@date(dd.MM.yyyy)@` becomes `23.02.2021`

Date reformatting

The special date-reformat placeholder can be used to interpret a date in one format and then have it printed in another format. The usage looks as follows:

```
@dateReformat(DATE|FROM_FORMAT|TO_FORMAT)@
```

NOTE: Don't put any spaces before or after the parentheses or the separating vertical bar symbols. (The spaces would be considered part of the corresponding parameter.)

This placeholder uses the vertical bar symbol instead of commas to separate its parameters, because the parameters themselves may contain commas.

The mechanism is as follows: the text found within the `DATE` parameter is interpreted in accordance to the format specification `FROM_FORMAT` so the software knows what date/time it represents, and it's then turned into `TO_FORMAT`.

The `FROM_FORMAT` and `TO_FORMAT` parameters are akin to the `FORMAT` parameter of the regular `date()` placeholder described in the previous section.

Following is an example for printing the expiry date of a batch number in a custom format, exploiting the fact that the placeholder `@expiryDateGS1@` will always produce a date of the format `yyMMdd`.

```
@dateReformat(@batchExpiryGS1@,yyMMdd,dd.MM.yyyy)@
```

JavaScript bindings

The HTML document resulting from the interpretation of a COBI.wms label template is rendered by a

full-fledged browser engine. This means that you can use not only CSS but even JavaScript in the template. However, this has to be enabled explicitly by placing the following pseudo-placeholder anywhere in the file, such as in an HTML comment somewhere at the top:

```
@useJavaScript@
```

NOTE: When JS is enabled, the actual printing of the rendered HTML document will not be triggered automatically anymore. Instead, you have to trigger the printing manually from your JS code by calling `cobiwms.print()` at some point.

During execution of your JS code, you have access to the special object `cobiwms` which contains a number of functions. Their explanation follows.

```
cobiwms.get(name)
```

The parameter `name` must be a string. It will be interpreted as a placeholder, and its value returned as a string. For example, calling `cobiwms.get("itemName")` will give you the item name as a string.

(You could also just use placeholders like `@itemName@` in your code, but then the value would be injected as-is into your code. That means: you would have to wrap it in quotation marks like "`@itemName@`", and any quotation marks appearing within the actual value would break the code. Therefore it's much better to use `cobiwms.get()` instead.)

```
cobiwms.prompt(title, callback)
```

Calling this function makes the app present a popup with an input field to the user. The parameter `title`, which must be a string, will be the title of the popup. The parameter `callback` must also be a string, and must represent a function. Said function will be called with one argument (a string representing the value entered by the user) after the user has confirmed their input.

Example usage:

```
window.handleCurrency = function (currency) {
    ...
}

cobiwms.prompt("Enter currency", "window.handleCurrency")
```

Example template files

The following example files serve to demonstrate the syntax and some of the features.

File 1

This is a very simple template demonstrating the use of placeholders to print the item code and name, price, and a timestamp.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <style>
        @page {
          /* You can use width/height, or a standardized size. */
          /* For example, the following two are equivalent: */
          /* size: 148mm 105mm; */
          /* size: A6 landscape; */
          size: A6 landscape;

          /* Don't change, use the container padding below. */
          margin: 0;
          padding: 0;
        }

        body {
          /* We could use 100vw/100vh to cover the whole size declared with
@page, but this doesn't
             work with ZPL printing, so repeat the page dimensions
explicitly.  Note that we can't
             use standardized sizes like A6 here.  The height should be
minimally reduced to make
             sure the HTML renderer doesn't start a second page. */
          width: 148mm;
          height: 104.9mm;

          /* Don't change, use the container below. */
          margin: 0;
          padding: 0;

          /* Useful to diagnose dimension issues. */
          outline: 0.5mm solid black;
          outline-offset: -0.5mm;
        }

        .container {
          /* Don't change. */
          box-sizing: border-box;
          position: relative;
          width: 100%;
          height: 100%;

          font-family: sans-serif;
          font-size: 18pt;

          /* Global padding from the edges. */
          padding: 3mm;
        }
    </style>
```

```
</head>
<body>
<div class="container">

    <div style="float: left;">
        <b>Code:</b> @itemCode@
    </div>

    <div style="float: right;">
        <b>Price:</b> @itemPriceCurrencySymbol(1)@
@itemPriceFormattedValue(1)@
    </div>

    <div style="text-align: center; margin-top: 24mm;">
        @itemName@
    </div>
    <div style="text-align: center; margin-top: 2mm;">
        @barcode(CODE_128,150,75,|90@itemCode@|10@batchNumber@)@
    </div>
    <div style="text-align: center; margin-top: 1mm; font-size: 0.8em;">
        (90)@itemCode@(10)@batchNumber@
    </div>

    <div style="position: absolute; left: 0; bottom: 0; padding: inherit;">
        <b>@date(yyyy-MM-dd HH:mm)@</b>
    </div>

    <div style="position: absolute; right: 0; bottom: 0; padding: inherit;">
        <b>COBI.wms Sample Label</b>
    </div>

</div>
</body>
</html>
```

File 2

This file demonstrates the use of user-input values and JavaScript integration.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8"/>
    <style>
        @page {
          /* You can use width/height, or a standardized size. */
          /* For example, the following two are equivalent: */
          /* size: 148mm 105mm; */
          /* size: A6 landscape; */
          size: A6 landscape;
```

```
          /* Don't change, use the container padding below. */
          margin: 0;
          padding: 0;
        }

        body {
          /* We could use 100vw/100vh to cover the whole size declared with
@page, but this doesn't
             work with ZPL printing, so repeat the page dimensions
explicitly.  Note that we can't
             use standardized sizes like A6 here.  The height should be
minimally reduced to make
             sure the HTML renderer doesn't start a second page. */
          width: 148mm;
          height: 104.9mm;

          /* Don't change, use the container below. */
          margin: 0;
          padding: 0;

          /* Useful to diagnose dimension issues. */
          outline: 0.5mm solid black;
          outline-offset: -0.5mm;
        }

        .container {
          /* Don't change. */
          box-sizing: border-box;
          position: relative;
          width: 100%;
          height: 100%;

          font-family: sans-serif;
          font-size: 18pt;

          /* Global padding from the edges. */
          padding: 3mm;
        }
    </style>
    <!-- Input definitions:

    @input(text[Insert value])@
    @input(selection[Select value];Value 1,Value 2,Value 3)@
    @input(codeFormat[Barcode type];CODE_128,DATA_MATRIX,QR_CODE)@

    -->

    <!-- @useJavaScript@ -->
    <script>
        window.onload = function() {
```

```
            var insert = document.getElementById('js-insert')
            insert.textContent = "Hello World!"
            cobiwms.print()
        }
    </script>
</head>
<body>
<div class="container">

    <div style="float: left;">
        <b>JS-Insert:</b> <span id="js-insert"></span>
    </div>

    <div style="float: right;">
        <b>Selection:</b> @input(selection)@
    </div>

    <div style="text-align: center; margin-top: 24mm;">
        @itemName@
    </div>
    <div style="text-align: center; margin-top: 2mm;">
        @barcode(@input(codeFormat)@,150,75,|90@itemCode@|10@batchNumber@)@
    </div>
    <div style="text-align: center; margin-top: 1mm; font-size: 0.8em;">
        (90)@itemCode@(10)@batchNumber@
    </div>

    <div style="position: absolute; left: 0; bottom: 0; padding: inherit;">
        <b>Input:</b> @input(text)@
    </div>

    <div style="position: absolute; right: 0; bottom: 0; padding: inherit;">
        <b>COBI.wms Sample Label</b>
    </div>

</div>
</body>
</html>
```