LabelCross Print

DEVELOPER REFERENCE

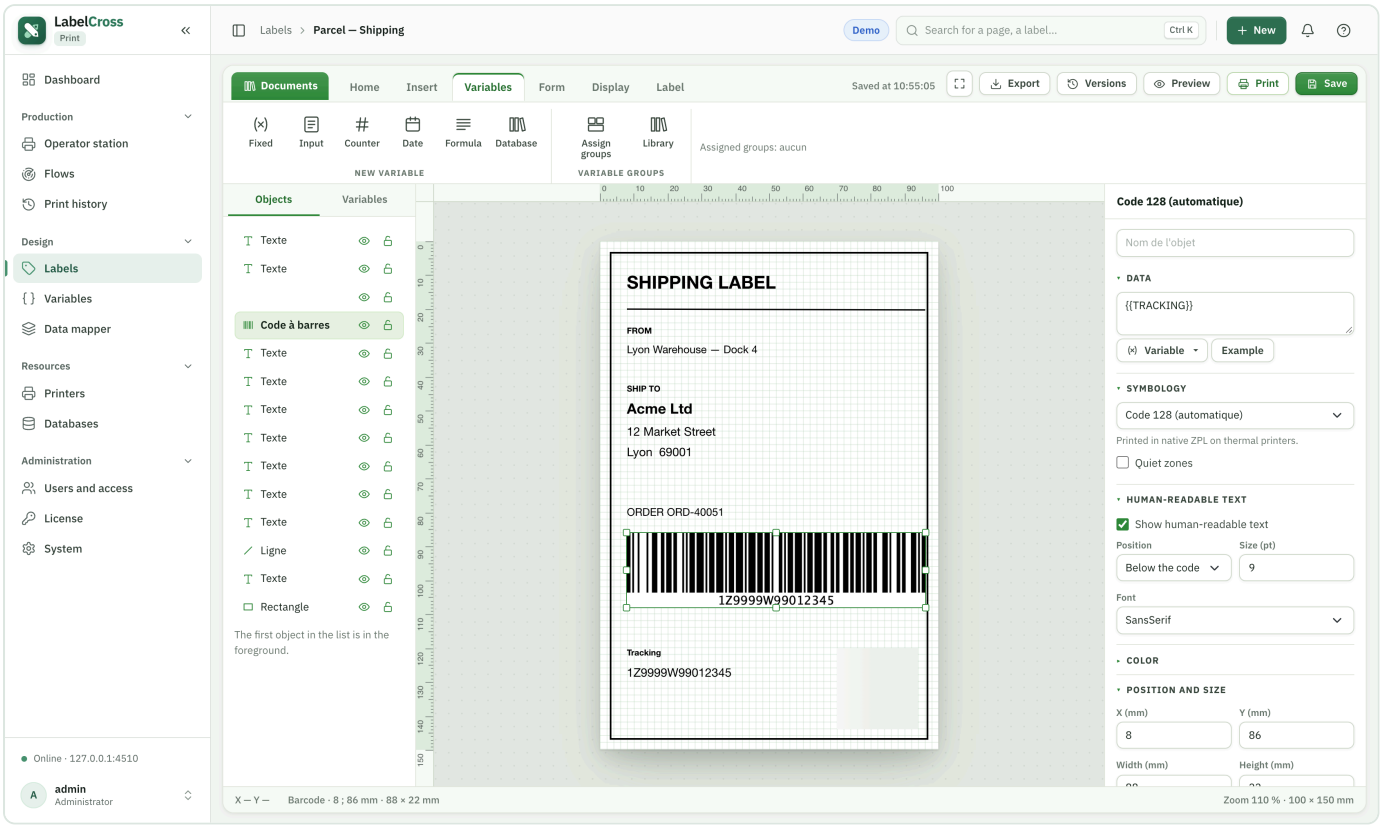
Variable formulas in the label designer

The complete reference for label developers: every function, operator and token, with a signature, an explanation and worked examples showing the exact result.

PART 1

Syntax essentials

A formula is plain text containing one or more **expressions** written between double braces `{{ }}`. Everything outside the braces is printed literally. Formulas can be used in the value of a text object, in a barcode's data, and in a **Calculated field**.



The Variables tab — where label variables are defined and used by formulas.

Variables

- Reference a variable by name: `{{Lot}}`, `{{TRACKING}}`.
- If the name contains spaces or symbols, wrap it in square brackets: `{{[Num-Lot]}}`, `{{[Order ref]}}`.
- Mix text and expressions freely: `Order {{ORDER}} — {{CITY}}`.

Function calls

- Arguments are separated by a **semicolon** `;` : `{{LEFT(Lot; 3)}}`.
- Function names are **case-insensitive** (`LEFT`, `Left`, `left` are equal). French names are accepted as aliases (see Part 5).
- Calls can be nested: `{{UPPER(LEFT(Name; 3))}}`.

Text literals

- Write text in double or single quotes: `"ABC"`, `'ABC'`.
- Escapes inside quotes: `\n` (new line), `\t` (tab), `\\` (backslash), `\uXXXX` and `\u{…}` (Unicode code point).

Operators

Operator	Meaning	Example → result
`&`	Concatenate text	`"REF-" & "42"` → **REF-42**
`+ − * /`	Arithmetic on numbers	`"7" / "2"` → **3.5**
`= <> < > <= >=`	Comparison → `1` (true) or `0` (false). Numeric if both sides are numbers, otherwise text order.	`"10" > "9"` → **1**

Values, numbers & truth

- **Numbers** accept a dot or a comma as decimal separator on input (`3.5` or `3,5`); results are written with a dot and up to 10 decimals, trailing zeros removed.
- **Truthiness** (for `IF`, `AND`, `OR`, `NOT`): empty, `0`, `false` and `faux` are *false*; everything else is *true*.
- **Empty & errors:** an empty value yields an empty result for date functions. Invalid data (a bad date or a non-number where a number is required) is an **error** and printing is refused — fix the source data or wrap it (e.g. `DEFAULT`, `ISEMPTY`).

Regions & calendars

Date and number functions take an optional region code as their last argument. It selects the week system and the language of month/day names:

Codes	Week system	Month / day names
`FR · ISO · EU · UK · GB · DE · (none)`	ISO 8601 — weeks start Monday; `WEEKDAY` 1 = Monday … 7 = Sunday.	French (e.g. *novembre*, *samedi*).
`EN · US · USA`	US — weeks start Sunday; `WEEKDAY` 1 = Sunday … 7 = Saturday.	English (e.g. *November*, *Saturday*).

Accepted date inputs

Functions that take a date accept these text formats automatically:

- **ISO**: `2026-11-14`, optionally with time `2026-11-14 08:30` or `2026-11-14T08:30:00`.
- **Day/Month/Year** with `/` `.` or `-` separators, 2- or 4-digit year: `14/11/2026`, `14-11-26`.
- **8 digits**: `20261114` (yyyymmdd).
- Any other layout: parse it explicitly with `DATEVALUE(text; "format")`.

Text functions

Signature	Description	Example → result
`LEFT(text; n)`	First `n` characters.	`LEFT("ABCDEF"; 3)` → **ABC**
`RIGHT(text; n)`	Last `n` characters.	`RIGHT("ABCDEF"; 2)` → **EF**
`MID(text; start; [len])`	Substring from 1-based `start`, `len` characters (to the end if omitted).	`MID("ABCDEF"; 2; 3)` → **BCD**
`LEN(text)`	Number of characters.	`LEN("Lot-12")` → **6**
`UPPER(text)`	Upper case.	`UPPER("ab-c")` → **AB-C**
`LOWER(text)`	Lower case.	`LOWER("AB-C")` → **ab-c**
`PROPER(text)`	Capitalise each word.	`PROPER("jean dupont")` → **Jean Dupont**
`TRIM(text)`	Remove leading/trailing spaces and collapse inner runs to one space.	`TRIM(" a b ")` → **a b**
`CONCAT(a; b; …)`	Join all arguments.	`CONCAT("REF-"; "42")` → **REF-42**
`REPLACE(text; find; by)`	Replace every occurrence of `find` with `by`.	`REPLACE("A-B-C"; "-"; "/")` → **A/B/C**
`PADLEFT(text; width; [fill="0"])`	Pad on the left to `width` with `fill`.	`PADLEFT("42"; 5)` → **00042**
`PADRIGHT(text; width; [fill=" "])`	Pad on the right to `width`.	`PADRIGHT("42"; 5; "0")` → **42000**
`REPEAT(text; n)`	Repeat the text `n` times (0–1000).	`REPEAT("AB"; 3)` → **ABABAB**
`REVERSE(text)`	Reverse the characters.	`REVERSE("ABC")` → **CBA**
`FIND(text; search)`	1-based position of the first occurrence, or `0` if not found.	`FIND("ABCDE"; "CD")` → **3**
`BEFORE(text; sep)`	The part before the first `sep` (the whole text if not found).	`BEFORE("name@host"; "@")` → **name**
`AFTER(text; sep)`	The part after the first `sep` (empty if not found).	`AFTER("name@host"; "@")` → **host**
`SPLIT(text; sep; index)`	The `index`-th piece (1-based; negative counts from the end).	`SPLIT("a,b,c"; ","; 2)` → **b**
`CHAR(code; …)`	Character(s) from Unicode code point(s).	`CHAR(72; 73)` → **HI**
`CODE(text)`	Unicode code point of the first character.	`CODE("A")` → **65**
`UNACCENT(text)`	Remove accents / diacritics.	`UNACCENT("Café")` → **Cafe**
`NORMALIZE(text; [form="NFC"])`	Unicode normalisation: NFC, NFD, NFKC or NFKD.	`NORMALIZE("ﬁ"; "NFKC")` → **fi**
`ONLYDIGITS(text)`	Keep digits only.	`ONLYDIGITS("AB12-3")` → **123**
`ONLYLETTERS(text)`	Keep letters only.	`ONLYLETTERS("AB12-3")` → **AB**
`ONLYALNUM(text)`	Keep letters and digits only.	`ONLYALNUM("AB 12-3!")` → **AB123**

Number functions

Signature	Description	Example → result

`ROUND(number; [decimals=0])`	Round half-up to `decimals` places; negative rounds to tens/hundreds.	`ROUND("3.14159"; 2)` → **3.14** `ROUND("1234"; -2)` → **1200**
`INT(number)`	Truncate toward zero (drop decimals).	`INT("3.99")` → **3**
`ABS(number)`	Absolute value.	`ABS("-5")` → **5**
`MOD(a; b)`	Remainder of `a ÷ b`.	`MOD("17"; "5")` → **2**
`MIN(a; …)`	Smallest number.	`MIN("3"; "7"; "1")` → **1**
`MAX(a; …)`	Largest number.	`MAX("3"; "7"; "1")` → **7**
`VALUE(text)`	Parse text to a number (empty stays empty).	`VALUE("0042")` → **42** `VALUE("3,5")` → **3.5**
`FORMATNUMBER(number; decimals; [region])`	Group thousands and set the decimals. FR uses a space + comma; US uses a comma + dot.	`FORMATNUMBER("1234567.5"; 2)` → **1 234 567,50** `…; 2; "US")` → **1,234,567.50**
`CHECKDIGIT(digits)`	The GS1 / EAN mod-10 check digit for the given digits.	`CHECKDIGIT("123456789012")` → **8**

Date & time functions

Signature	Description	Example → result
`TODAY()`	Today's date (no time).	`TODAY()` → **2026-11-14**
`NOW()`	Current date and time.	`NOW()` → **2026-11-14T08:30:00**
`DATEVALUE(text; [format])`	Parse text to an ISO date; use `format` for non-standard inputs.	`DATEVALUE("14/11/2026")` → **2026-11-14** `DATEVALUE("261114"; "yyMMdd")` → **2026-11-14**
`FORMATDATE(date; format; [region])`	Format a date with the tokens below.	`FORMATDATE("2026-11-14"; "dd/MM/yyyy")` → **14/11/2026**
`DAY(date)`	Day of the month.	`DAY("2026-11-14")` → **14**
`MONTH(date)`	Month number.	`MONTH("2026-11-14")` → **11**
`YEAR(date)`	4-digit year.	`YEAR("2026-11-14")` → **2026**
`YEAR2(date)`	2-digit year.	`YEAR2("2026-11-14")` → **26**
`WEEK(date; [region])`	Week number (ISO by default, US with a US region).	`WEEK("2026-01-05")` → **2**
`ISOYEAR(date)`	The ISO week-numbering year (differs from YEAR near Jan 1).	`ISOYEAR("2027-01-01")` → **2026**
`WEEKDAY(date; [region])`	Day of week. ISO: 1=Mon … 7=Sun. US: 1=Sun … 7=Sat.	`WEEKDAY("2026-11-14")` → **6**
`DAYOFYEAR(date)`	Day of the year (1–366).	`DAYOFYEAR("2026-01-10")` → **10**
`JULIAN(date)`	Julian code: 2-digit year + 3-digit day-of-year.	`JULIAN("2026-01-10")` → **26010**
`DAYNAME(date; [region])`	Weekday name (French by default, English for a US region).	`DAYNAME("2026-11-14"; "US")` → **Saturday**
`MONTHNAME(date; [region])`	Month name (French by default, English for a US region).	`MONTHNAME("2026-11-14"; "US")` → **November**
`ADDDAYS(date; n)`	Shift by `n` days (negative to go back).	`ADDDAYS("2026-11-14"; -14)` → **2026-10-31**
`ADDMONTHS(date; n)`	Shift by `n` months.	`ADDMONTHS("2026-11-14"; 2)` → **2027-01-14**
`ADDYEARS(date; n)`	Shift by `n` years.	`ADDYEARS("2026-11-14"; 1)` → **2027-11-14**
`DATEDIFF(date1; date2)`	Whole days from `date1` to `date2`.	`DATEDIFF("2026-11-14"; "2026-11-24")` → **10**

Date format tokens (for `FORMATDATE`)

Any character that is not a token is printed as-is; wrap literal letters in single quotes (`'…'`; `''` prints a quote).

Token	Meaning	Token	Meaning
`yyyy / yy`	Year, 4 / 2 digits	`EEEE / EEE`	Weekday name, full / short
`MM / M`	Month, 2 digits / minimal	`HH / H`	Hour, 2 digits / minimal
`MMMM / MMM`	Month name, full / short	`mm / ss`	Minutes / seconds

`dd / d`	Day, 2 digits / minimal	`ww / w`	Week number, 2 digits / minimal
`DDD / D`	Day-of-year, 3 digits / minimal	`u`	Weekday number

Parse tokens for `DATEVALUE(text; "format")` are the numeric ones only: `yyyy`, `yy`, `MM`, `M`, `dd`, `d`, `HH`, `H`, `mm`, `ss`.

Logic functions

Signature	Description	Example → result
`IF(condition; then; [else])`	Return `then` if the condition is true, otherwise `else` (empty if omitted). Only the chosen branch is evaluated.	`IF("5" > "3"; "OK"; "NO")` → **`OK`**
`AND(a; …)`	`1` if every argument is true, else `0`.	`AND("1"; "1"; "0")` → **`0`**
`OR(a; …)`	`1` if any argument is true, else `0`.	`OR("0"; "1")` → **`1`**
`NOT(x)`	Logical negation.	`NOT("0")` → **`1`**
`ISEMPTY(x)`	`1` if the value is empty/blank, else `0`.	`ISEMPTY("")` → **`1`**
`DEFAULT(x; fallback)`	Return `x`, or `fallback` when `x` is empty.	`DEFAULT(""; "N/A")` → **`N/A`**

French aliases

These French names are accepted and behave exactly like their English function:

Alias	Function	Alias	Function	Alias	Function
`GAUCHE`	LEFT	`DROITE`	RIGHT	`STXT`	MID
`NBCAR`	LEN	`MAJUSCULE`	UPPER	`MINUSCULE`	LOWER
`NOMPROPRE`	PROPER	`SUPPRESPACE`	TRIM	`CONCATENER`	CONCAT
`SUBSTITUE / SUBSTITUTE`	REPLACE	`CAR`	CHAR	`CNUM`	VALUE
`AUJOURDHUI`	TODAY	`MAINTENANT`	NOW	`TEXTE_DATE`	FORMATDATE
`JOUR`	DAY	`MOIS`	MONTH	`ANNEE`	YEAR
`SEMAINE / NO_SEMAINE`	WEEK	`JOURSEM`	WEEKDAY	`QUANTIEME`	DAYOFYEAR
`ARRONDI`	ROUND	`ENT`	INT	`SI`	IF
`ET`	AND	`OU`	OR	`NON`	NOT
`ESTVIDE`	ISEMPTY				

PART 7

Worked examples (recipes)

Realistic formulas combining several functions. Assume variables such as `SEQ`, `GTIN12`, `Name`, `Batch`, `ProdDate` and `Price`.

Goal	Formula → result
Zero-padded sequence number	`{{PADLEFT(SEQ; 6)}}` → **`000042`**
GTIN-13 with its check digit	`{{CONCAT(GTIN12; CHECKDIGIT(GTIN12))}}` → **`1234567890128`**
Clean SKU from a product name	`{{ONLYALNUM(UPPER(UNACCENT(Name)))}}` `Name = "Café Noël"` → **`CAFENOEL`**
Lot code: year + week	`{{"LOT-" & YEAR2(TODAY()) & "W" & PADLEFT(WEEK(TODAY()); 2)}}` → **`LOT-26W46`**
Best-before = production + 18 months	`{{FORMATDATE(ADDMONTHS(ProdDate; 18); "dd/MM/yyyy")}}` `ProdDate = 2026-11-14` → **`14/05/2028`**
Julian production date	`{{JULIAN(TODAY())}}` → **`26318`**
Fallback when a field is empty	`{{IF(ISEMPTY(Batch); "NO BATCH"; Batch)}}`
Price with currency (French format)	`{{FORMATNUMBER(Price; 2) & " €"}}` `Price = 1299.9` → **`1 299,90 €`**

💡 Always **preview** a calculated value with representative example data before validating the label. If a formula can receive empty or malformed input, guard it with `ISEMPTY` or `DEFAULT` so printing is never blocked unexpectedly.