Repeatable Group Field

User Guide · RSForm! Pro Extension · Version 1.8.3

What Is This Plugin?

Repeatable Group Field adds dynamic, repeatable row groups to RSForm! Pro forms. Without this plugin, RSForm! Pro has no built-in way to collect a variable number of structured entries in a single form submission — for example, registering multiple attendees, listing several products in an order, or recording multiple work history entries.

The plugin introduces two new field types — **Repeatable Group: Start** and **Repeatable Group: End** — that act as opening and closing markers around any set of fields. Everything between these two markers becomes a repeatable unit. Users filling in the form can add or remove rows as needed.

Beyond the original add/remove mechanics, the plugin gives you fine-grained control over **how the group looks and behaves on the frontend** (layout, field labels, where the "Add row" control appears), **how submitted data is rendered in emails and PDF attachments** (plain lists, transposed tables, field-major tables, card views — with or without inline styling), and **how submissions are exported** from the RSForm! Pro backend. Since version 1.7.0, groups can also contain **File Upload** fields, with files stored separately for each row.

Requirements

Requirement	Minimum version
RSForm! Pro	Any current release
PHP	7.4 (PHP 8.4 compatible)
PHP ZipArchive	Required for XLSX export

Installation

1. Log in to your Joomla backend.
2. Go to **Extensions** → **Manage** → **Install**.
3. On the **Upload Package File** tab, select the `plg_system_repeatablegroup.zip` file and click **Upload & Install**.
4. Go to **Extensions** → **Plugins**.
5. Search for **"Repeatable Group"** and set the plugin status to **Enabled**.

The plugin is now active. Two new field types will appear in the RSForm! Pro form editor under the **Advanced** category:

- **Repeatable Group: Start**
- **Repeatable Group: End**

> **Updating from an earlier version?** Re-running the installer over an existing installation upgrades the plugin in place. The installer's update script automatically applies any new database properties introduced since your current version — no manual SQL needed. After updating, clear the Joomla cache (System → Clear Cache) and reload the submissions view to see the new RG Export button.

Setting Up a Repeatable Group

Step 1 — Add the Start marker

In the RSForm! Pro form editor, add a **Repeatable Group: Start** field at the point where the repeatable section should begin.

(1.8.0) Every property of the three field types has a help text: hover the property's label in the field editor to read it.

Configure the following properties:

Property	Description
Field Name	A unique identifier for this field (required by RSForm! Pro).
Group ID	Optional. A short identifier used in email/PDF placeholders (e.g. `attendees`). If left empty, the Field Name is used. Letters, numbers, and underscores only.
Caption	Optional heading displayed above the group.
Minimum rows	The minimum number of rows the user must fill in. Default: 1.
Maximum rows	The maximum number of rows allowed. Set to `0` for unlimited.
Add button text	Label for the button that adds a new row. Leave empty to use the default ("Add row").
Remove button text	Label for the remove button on each row. Leave empty to use the default ("Remove").
Show row	Whether to display a numbered header on each row.

number	
Row label	Text shown before the row number (e.g. "Participant"). Only used if **Show row number** is enabled.
Field layout	Controls how the fields inside each row are arranged: stacked, side by side, or a styled "card" layout. See Field Layout.
Show field labels	Controls when and where each field's label is shown — every row, once as a header, only on the first row, or never. See Field Labels.
"Add row" button position	Controls where the control that adds a new row appears: below the group, above it, or as a small "+" button in every row. See Add Button Position.
Custom validation error message	Optional. Overrides the default required-field error message for all fields in this group. Leave empty to use the default messages.
Show duplicate button	When enabled, each row gets a **duplicate** button that clones the row — including all current values — and inserts the copy immediately below. Useful for registering similar entries quickly.
Sortable rows	When enabled, rows can be reordered by drag-and-drop using the handle (⠿) on each row header.

Step 2 — Add fields inside the group

Place any supported fields between the Start and End markers. These fields will be repeated for every row the user adds.

> **Important:** The form editor shows a compatibility indicator next to each field inside a repeatable group. Check the Field Compatibility section below before placing fields.

> **Grid layout:** the group is defined by the field **order** between the Start and End markers, not by the grid. The fields can sit in one grid column or in separate grid rows. Since 1.7.9 the plugin collects them on the page and shows the group where the Start marker is. Keep the Start and End markers on the **same form page** when you use page breaks.

Step 3 — Add the End marker

Add a **Repeatable Group: End** field immediately after the last field in the group. No configuration is required — the End marker simply closes the group — but it also has one optional property:

Property	Description
Summary template	Free-form HTML, rendered below the rows, where `{...}` blocks compute live totals (sums, averages, row counts, etc.) from the group's data. See Summary Template — Live Calculated Totals. Leave empty for no summary.

Customising the Group's Appearance and Behaviour

Field Layout

The **Field layout** property controls how the fields inside each row are arranged.

Value	Result
VERTICAL *(default)*	Fields are stacked below each other, exactly as RSForm! Pro renders them normally. Best for rows with many fields, long labels, or larger inputs such as textareas and checkbox groups.
HORIZONTAL	Fields in a row are placed side by side, with minimal built-in styling: the label sits above its input, and inputs stretch to fill the available space. This is a deliberately plain "skeleton" layout, meant as a starting point for your own CSS.
HORIZONTAL_STYLED	The same side-by-side arrangement as HORIZONTAL, but with a polished, ready-to-use card style out of the box: rounded corners, a subtle shadow, a coloured accent border, compact uppercase labels, and inputs with a focus highlight.

> On screens narrower than 576px, both HORIZONTAL and HORIZONTAL_STYLED automatically fall back to a stacked layout.

Field Labels

The **Show field labels** property controls when each field's label/caption is displayed.

Value	Result
ALL_ROWS *(default)*	Every row shows its own field labels — the normal RSForm! Pro behaviour.
HEADER_ROW	A single header row, built from each field's **Caption**, is shown once at the top of the group. Per-row labels are hidden in every row. Works best with HORIZONTAL or HORIZONTAL_STYLED layout.

FIRST_ROW_ONLY	Only the first row shows field labels. Rows added afterwards don't repeat them.
HIDDEN	Field labels are never shown, in any row.

Add Button Position

Value	Result
BOTTOM *(default)*	The "Add row" button appears below the group, after the last row.
TOP	The "Add row" button is moved to the top of the group, before the first row.
EACH_ROW	The main "Add row" button is hidden. Instead, every row's header gets a small **"+"** button next to the **"×"** (remove) button. Clicking it inserts a new, empty row **immediately below that row** — not at the end of the list.

> All three options respect **Minimum rows** and **Maximum rows**.

Summary Template — Live Calculated Totals

The End marker's **Summary template** property lets you display calculated totals below the group, with **no custom development**. It's free-form HTML containing one or more `{...}` expressions.

Available functions

Function	Result
`count`	The number of filled-in rows.
`sum <fieldname>`	The sum of `<fieldname>`'s values across all rows.
`avg <fieldname>`	The average of `<fieldname>`'s values across all rows.
`min <fieldname>` / `max <fieldname>`	The smallest / largest value of `<fieldname>` across all rows.

Arithmetic and rounding

Inside `{...}`, the result can be combined with ordinary arithmetic (`+ - * /` and parentheses) and rounded with `| N` (N decimal places):

```
<p>Subtotal: {sum total} Ft</p>
<p><strong>Total incl. VAT (27%): {sum total * 1.27 | 2} Ft</strong></p>
```

Math functions

The same functions as in Computed Fields are available inside `{...}`: `round`, `floor`, `ceil`, `abs`, `min`, `max`. Arguments are separated by a **semicolon** (`;`). See Functions for details.

With three rows where `total` is 381, 127 and 254, and `qty` is 3, 1 and 2:

Expression	Output
`{round(sum total * 1,27; 0)}`	`968`
`{max(sum total; 1000)}`	`1000`
`{ceil(sum qty / 4)}`	`2`
`{round(avg total * 1,27; 2) | 2}`	`322.58`

The same totals appear in confirmation emails and PDFs via `{groupId:summary}`.

Computed Field — Per-Row Live Calculations

In addition to the Summary template (which aggregates across all rows), the plugin provides a **Repeatable Group: Computed Field** — a dedicated field type that calculates a value *within a single row* based on other fields in the same row. The result is written into the field in real time as the user types, and is stored in the database on submission.

Adding a Computed Field

In the RSForm! Pro form editor, add a **Repeatable Group: Computed Field** between the Start and End markers. It behaves like any other grouped field — it can be placed anywhere in the row.

Property	Description
Name	The field's technical name, used in formulas and placeholders.
Caption	Optional label shown above the field.
Formula (simple mode)	The calculation expression used in simple mode. See syntax below.
Calculation mode	**Simple** (default) — a single arithmetic expression. **Conditional (if/then/else)** — a condition decides which of two expressions is evaluated. **Multi-branch (if / else if /

	else) — any number of *condition* => *result* rules; the first matching rule wins. **Text** — joins field values into text, without calculation (e.g. a full name).
Condition (conditional mode)	The comparison expression, e.g. `{qty} > 0` or `{qty} * {price} >= 1000`. See Condition syntax. Active in conditional mode only.
If true — then (conditional mode)	The expression evaluated when the condition is met. Same syntax as the simple formula. Active in conditional mode only.
If false — else (conditional mode)	The expression evaluated when the condition is not met. Same syntax as the simple formula. Active in conditional mode only.
Branches (multi-branch mode)	One rule per line: `condition => result`. See Multi-branch mode. Active in multi-branch mode only.
Text template (text mode)	Text with `{fieldname}` references, e.g. `{title} {first_name} {last_name}`. See Text mode. Active in text mode only.
Read only	If set to **YES**, the user cannot edit the computed value. Recommended for totals and derived values.
Hidden	If set to **YES**, the field is invisible on the frontend but its value is still calculated and stored on submission. Useful for intermediate values referenced by other fields or the Summary template.
Display as	**Input field** (default): a text field. **Plain text**: the result as text, not a field. **Badge**: the result as a small label that can be coloured per value. See Display as — text or badge. Ignored if **Hidden** is YES.

> **All formula properties are always visible** in the field editor — the RSForm! Pro property panel is a static form with no show/hide capability. The property labels indicate which mode each field belongs to. Only the properties matching the active **Calculation mode** are evaluated; the others are stored but ignored. The one exception is multi-branch mode, which falls back to other fields if **Branches** is empty — see If Branches is empty.

Simple mode — formula syntax

A formula references other fields in the **same row** using `{fieldname}` syntax and combines them with ordinary arithmetic. The field shows the result as the user types.

Operators

Operator	Meaning	Example formula	Row values	Result
`+`	addition	`{qty} + {price}`	qty = 3, price = 100	`103`
`-`	subtraction	`{income} - {cost}`	income = 5000, cost = 3200	`1800`
`*`	multiplication	`{qty} * {price}`	qty = 3, price = 100	`300`
`/`	division	`{total} / {people}`	total = 1000, people = 3	`333.333333`
`( )`	grouping	`({a} + {b}) / 2`	a = 4, b = 5	`4.5`
`**`	power	`{side} ** 2`	side = 4	`16`
`-` (sign)	negative value	`-{a} + {b}`	a = 4, b = 5	`1`

Numbers can appear anywhere in the expression, including at the start: `100 - {a}` → `96`, `2 * {a}` → `8` (a = 4).

Without rounding, a result shows up to six decimals, with trailing zeros removed.

Rounding — `| N`

Add `|` and a number at the end of the expression to round the result to that many decimals.

Example formula	Row values	Result
`{total} / {people} | 2`	total = 1000, people = 3	`333.33`
`{total} / {people} | 0`	total = 1000, people = 3	`333`
`{qty} * {price} | 2`	qty = 3, price = 100	`300.00`
`{net} * (1 + {vat} / 100) | 2`	net = 300, vat = 27	`381.00`

Functions

Function	Result	Example formula	Row values	Result
`round(`	`x` rounded to `n` decimals; `n` may be	`round({net} *`	net = 1234.5	`1568`

`x; n)`	omitted (0) or negative (tens, hundreds)	`1,27; 0)`		
		`round({net} * 1,27; -2)`	net = 1234.5	`1600`
		`round({price} * 0,9; 1)`	price = 1999	`1799.1`
`floor(x)`	rounded down to a whole number	`floor({minutes} / 60) h`	minutes = 135	`2 h`
`ceil(x)`	rounded up to a whole number	`ceil({qty} / {per_box})`	qty = 23, per_box = 10	`3`
		`ceil({net} / 100) * 100`	net = 1234.5	`1300`
`abs(x)`	absolute value	`abs({planned} - {actual})`	planned = 40, actual = 46	`6`
`min(a; b; …)`	the smallest argument	`min({qty} * {price}; 50000)`	qty = 12, price = 4990	`50000`
`max(a; b; …)`	the largest argument	`max(0; {paid} - {due})`	paid = 8000, due = 10000	`0`
		`max({a}; {b}; {c})`	a = 3, b = 9, c = 5	`9`

- **Arguments are separated by a semicolon** (`;`), because the comma is the decimal separator: `round({x}; 2)`. Writing `round({x}, 2)` gives an empty field.
- Function names are not case-sensitive (`MAX` = `max`). An unknown function name (e.g. `sqrt({a})`) gives an empty field.
- Functions can be nested and combined with everything else: `Boxes: ceil({qty} / {per_box}) pcs` → `Boxes: 3 pcs`.
- `round(x; n)` rounds halves away from zero (`round(2.5)` = 3, `round(-2.5)` = −3).

`round()` or `| N`? `round()` changes the **value**; `| N` only changes how the result is **displayed**. With x = `2,345`:

Formula	Displayed	Numeric value
`{x} | 2`	`2.35`	`2.35`

`round({x}; 2)`	`2.35`	`2.35`
`round({x}; 0) | 2`	`2.00`	`2`

Use `round()` when the rounded value should be used further (in a chained field, a condition, or the Summary), and `| N` for a fixed number of decimals on screen.

Text before and after the result — prefix and suffix

Text before the expression is shown in front of the result (**prefix**). Text after the expression, or after `| N`, is shown behind it (**suffix**).

Example formula	Row values	Result
`{qty} * {price} EUR`	qty = 3, price = 100	`300 EUR`
`{qty} * {price} | 2 EUR`	qty = 3, price = 100	`300.00 EUR`
`Total: {qty} * {price}`	qty = 3, price = 100	`Total: 300`
`Amount: {qty} * {price} | 2 Ft`	qty = 3, price = 100	`Amount: 300.00 Ft`
`Avg: ({a} + {b}) / 2 | 1`	a = 4, b = 5	`Avg: 4.5`
`Item 3: {a} * {b}`	a = 4, b = 5	`Item 3: 20`

Prefix and suffix are display text only: the stored numeric value of `Amount: 300.00 Ft` is `300`. See Chained computed fields.

> **Tip:** Keep the calculation in one piece, from the first `{` or number to the last `}` or number. Text *between* field references (e.g. `{a} m × {b}`) is not supported, and the field then shows the formula itself instead of a result.

Input values

Situation	Example formula	Row values	Result
Decimal comma in the input	`{qty} * {price} | 0 Ft`	qty = `2,5`, price = 1200	`3000 Ft`
Empty field (counts as 0)	`{qty} * {price}`	qty = *(empty)*, price = 1200	`0`
Checkbox group (checked values are added up)	`{base} + {extras} | 0 Ft`	base = 10000, extras = ☑ 1500 ☑ 2500	`14000 Ft`
Division by zero	`{total} / {people}`	total = 1000, people = 0	*(empty)*

For Dropdown, Radio, and Checkbox fields, the option **value** (not its label) is used, so give those options numeric values.

Everyday examples

Purpose	Formula	Row values	Result
Line total	`{qty} * {price} | 2`	qty = 3, price = 100	`300.00`
Gross price from net + VAT %	`{net} * (1 + {vat} / 100) | 2`	net = 300, vat = 27	`381.00`
Net price from gross	`{gross} / (1 + {vat} / 100) | 2`	gross = 12700, vat = 27	`10000.00`
VAT amount from gross	`{gross} - {gross} / (1 + {vat} / 100) | 2`	gross = 12700, vat = 27	`2700.00`
Discounted line total	`{qty} * {price} * (1 - {discount} / 100) | 2`	qty = 3, price = 100, discount = 10	`270.00`
Area	`{w} * {h} | 2 m²`	w = `3,2`, h = `2,5`	`8.00 m²`
Percentage share	`{part} / {whole} * 100 | 1 %`	part = 45, whole = 180	`25.0 %`
Average of two fields	`({a} + {b}) / 2 | 1`	a = 4, b = 5	`4.5`
Unit conversion (km → miles)	`{km} * 0.621371 | 1 mi`	km = 42.195	`26.2 mi`
Unit conversion (°C → °F)	`{celsius} * 9 / 5 + 32 | 1 °F`	celsius = 22	`71.6 °F`

Conditional mode — formula syntax

When **Calculation mode** is set to **Conditional**, the field checks the **Condition** first. If it is true, the **If true** expression is shown; otherwise, the **If false** expression.

Both branches use the simple formula syntax: calculations, `| N`, prefix and suffix. A branch can also be plain text (`Passed`, `N/A`) or a fixed number (`0`, `5000 Ft`). The condition syntax is described in Condition syntax.

Examples

Discount only when one is given

Property	Value
Condition	`{discount} > 0`
If true	`{qty} * {price} * (1 - {discount} / 100) | 2`
If false	`{qty} * {price} | 2`

qty	price	discount	Result
3	100	10	`270.00`
3	100	0	`300.00`

Free shipping above an order value

Property	Value
Condition	`{qty} * {price} >= 20000`
If true	`Free shipping`
If false	`1990 Ft`

qty	price	Result
5	4990	`Free shipping`
1	4990	`1990 Ft`

Minimum fee

Property	Value
Condition	`{hours} * {rate} < 5000`
If true	`5000 Ft`
If false	`{hours} * {rate} | 0 Ft`

hours	rate	Result
1	3500	`5000 Ft`
3	3500	`10500 Ft`

Pass / fail label

Property	Value
Condition	`{score} >= 50`
If true	`Passed`
If false	`Failed`

score	Result
72	`Passed`
38	`Failed`

Weekend surcharge from a checkbox (checkbox `weekend` with value `1`)

Property	Value
Condition	`{weekend} == 1`
If true	`{hours} * {rate} * 1.5 | 0 Ft`
If false	`{hours} * {rate} | 0 Ft`

weekend	hours	rate	Result
☑	4	3000	`18000 Ft`
☐	4	3000	`12000 Ft`

Two conditions together (`&&`): 10 % off for members ordering at least 10 items (`member` is a checkbox with value `1`)

Property	Value
Condition	`{member} == 1 && {qty} >= 10`
If true	`{qty} * {price} * 0.9 | 0 Ft`
If false	`{qty} * {price} | 0 Ft`

member	qty	price	Result
☑	12	1000	`10800 Ft`
☑	5	1000	`5000 Ft`

☐	12	1000	`12000 Ft`

Either condition (`||`)

Property	Value
Condition	`{qty} >= 100 || {vip} == 1`
If true	`Priority`
If false	`Standard`

qty	vip	Result
150	☐	`Priority`
20	☑	`Priority`
20	☐	`Standard`

Avoiding division by zero

Property	Value
Condition	`{people} > 0`
If true	`{total} / {people} | 0 Ft`
If false	`N/A`

total	people	Result
10000	4	`2500 Ft`
10000	0	`N/A`

Condition syntax

The same condition syntax is used in the **Condition** property (conditional mode) and on the left side of every rule in **Branches** (multi-branch mode).

A condition compares two arithmetic expressions. Each side can be a field reference, a number, or a calculation built from them with `+ - * /` and parentheses.

Comparison operators

Operator	Meaning	Example	Row values	Result

`>`	greater than	`{qty} > 0`	qty = 3	true
			qty = 0	false
`<`	less than	`{qty} < 10`	qty = 4	true
			qty = 12	false
`>=`	greater than or equal	`{qty} * {price} >= 1000`	qty = 4, price = 250	true
			qty = 3, price = 250	false
`<=`	less than or equal	`{probability} * {severity} <= 8`	probability = 2, severity = 4	true
			probability = 3, severity = 3	false
`==` or `=`	equal	`{type} == 2` (or `{type} = 2`)	type = 2	true
			type = 3	false
`!=`	not equal	`{qty} != 0`	qty = 5	true
			qty = *(empty)*	false

Combining comparisons

Combinator	Meaning	Example	Row values	Result
`&&`	and — both must be true	`{score} >= 9 && {score} <= 14`	score = 12	true
			score = 15	false
`||`	or — at least one must be true	`{qty} > 100 || {vip} == 1`	qty = 150, vip = ☐	true
			qty = 20, vip = ☑	true
			qty = 20, vip = ☐	false
`( … )`	grouping	`({a} > 1 || {b} > 2) && {c} == 0`	a = 3, b = 0, c = 0	true
			a = 3, b = 0,	false

			c = 1	

`&&` binds more tightly than `||`, as in most programming languages. `{a} > 1 || {b} > 0 && {c} > 0` means `{a} > 1 || ({b} > 0 && {c} > 0)`, so with a = 0, b = 1, c = 1 it is **true**. Use parentheses whenever the grouping should be different.

Functions in conditions

Both sides of a comparison can use functions:

Condition	If true	If false	Row values	Result
`abs({planned} - {actual}) <= 5`	`On target`	`Deviation`	planned = 40, actual = 43	`On target`
			planned = 40, actual = 46	`Deviation`

Text comparison

To check which option was chosen in a Dropdown, Radio or Checkbox field, or what was typed into a text field, compare the field with a text in quotes:

Condition	Row values	Result
`{status} == "paid"`	status = paid	true
`{status} = "PAID"`	status = paid	true
`{status} != "paid"`	status = open	true
`{status} == ""`	status = *(nothing selected)*	true
`{country} == "HU"`	country = ◉ HU	true
`{extras} == "breakfast"`	extras = ☑ wifi ☑ breakfast	true
`{extras} == "breakfast"`	extras = ☑ wifi	false
`{extras} != "breakfast"`	extras = ☑ wifi	true
`{risk_level} == "high"`	risk_level (Computed Field) shows `high`	true

- **The option value is compared, not its label.** A Dropdown option with the value `paid` and the label *Paid in full* matches `"paid"`, not `"Paid in full"`.
- Letter case and leading or trailing spaces do not matter: `"PAID"` = `"paid"`.
- Quotes can be `"…"`, `'…'`, or the Hungarian `„…”`.
- For **Checkbox groups** and multi-select Dropdowns, `==` is true if **any** checked value matches, and

`!=` is true if none does.

- For a **Computed Field**, the displayed text is compared, e.g. `high` (also with `[3] high`).
- Only `==` (or `=`) and `!=` work with text. `{status} > "a"` is always false.

Text comparisons combine freely with numeric ones:

Condition	Row values	Result
`{status} == "paid" && {qty} >= 5`	status = paid, qty = 5	true
	status = paid, qty = 2	false
`({status} == "paid" || {country} == "AT") && {qty} == 0`	status = open, country = AT, qty = 0	true
	status = open, country = HU, qty = 0	false

Examples in the other modes

Mode	Setup	Row values	Result
Conditional	Condition `{status} == "paid"`, If true `Paid`, If false `Unpaid`	status = paid	`Paid`
		status = open	`Unpaid`
Conditional	Condition `{extras} == "parking"`, If true `2000 Ft`, If false `0 Ft`	extras = ☑ parking ☑ wifi	`2000 Ft`
Conditional	Condition `{status} != "paid"`, If true `{qty} * 1500 | 0 Ft`, If false `0 Ft`	status = open, qty = 3	`4500 Ft`
		status = paid, qty = 3	`0 Ft`

Multi-branch, shipping fee by country:

```
{country} == "HU" => 1490 Ft
{country} == "AT" || {country} == "DE" => 3990 Ft
else => Contact us
```

country	Result
HU	`1490 Ft`

AT	`3990 Ft`
(nothing selected)	`Contact us`

Numbers, empty fields, and errors

- Both decimal point and decimal comma work, in the condition and in the field values: `{price} >= 1.5` is true for price = `1,5`.
- An empty field counts as `0`: `{qty} != 0` is false while qty is empty.
- A condition that cannot be evaluated counts as **false**, e.g. a typo like `{qty} >> 2` or an unclosed parenthesis like `({qty} > 2`.

> Without quotes, a condition compares **numbers**: `{type} == 2` uses the numeric option value. With quotes, it compares **text**; see Text comparison.

Multi-branch mode — if / else if / else

When a result depends on more than two cases (for example, turning a score into *low / medium / high*), set **Calculation mode** to **Multi-branch** and write one rule per line in **Branches**:

```
condition => result
```

The rules are checked from top to bottom, and the **first rule whose condition is true** decides the result. A line starting with `else =>` has no condition and always matches, so put it last as the default. If no rule matches and there is no `else` line, the field stays empty.

Results use the simple formula syntax: plain text (`low`), a number (`1490 Ft`), a calculation (`{qty} * {price} * 0.9 | 2`), or a calculation with text around it (`{done} / {total} * 100 | 0 % (on track)`).

Format rules:

- Empty lines and lines starting with `//` are ignored, so you can comment your rules.
- The separator is `=>`. It is never confused with the `>=` and `<=` comparison operators.
- A line without `=>` is skipped.

Examples

Risk matrix: probability × severity, both 1–5

```
// Risk level
{probability} * {severity} <= 8 => low
{probability} * {severity} <= 14 => medium
else => high
```


probability	severity	Score	Result
2	4	8	`low`
3	3	9	`medium`
2	7	14	`medium`
3	5	15	`high`
5	5	25	`high`

Because the first match wins, the second rule does not need `>= 9`: a score of 8 or less has already matched the first line. To write a closed range in full, use `&&`: `{score} >= 9 && {score} <= 14 => medium`.

Shipping fee by weight: fixed prices, with a text default

```
{weight} <= 2 => 1490 Ft
{weight} <= 10 => 2490 Ft
{weight} <= 30 => 4990 Ft
else => Contact us
```

weight	Result
1,5	`1490 Ft`
2	`1490 Ft`
7	`2490 Ft`
25	`4990 Ft`
45	`Contact us`

Quantity discount tiers: a different calculation in each branch

```
{qty} >= 100 => {qty} * {price} * 0.85 | 2
{qty} >= 50 => {qty} * {price} * 0.90 | 2
{qty} >= 10 => {qty} * {price} * 0.95 | 2
else => {qty} * {price} | 2
```

qty	price	Discount	Result
5	200	—	`1000.00`
10	200	5 %	`1900.00`

60	200	10 %	`10800.00`
120	200	15 %	`20400.00`

Put the highest tier first. In the opposite order, every quantity of 10 or more would stop at the 5 % rule.

Grade from points

```
{points} >= 90 => A
{points} >= 75 => B
{points} >= 60 => C
{points} >= 50 => D
else => F
```

points	Result
95	`A`
90	`A`
82	`B`
61	`C`
50	`D`
49	`F`

Size category from two fields (`&&`)

```
{w} <= 60 && {h} <= 40 => S
{w} <= 120 && {h} <= 80 => M
else => L
```

w	h	Result
50	30	`S`
50	70	`M`
100	60	`M`
130	60	`L`

Progress with a calculated result and text

```
{done} / {total} * 100 >= 100 => Done
{done} / {total} * 100 >= 50 => {done} / {total} * 100 | 0 % (on track)
else => {done} / {total} * 100 | 0 % (behind)
```

done	total	Result
10	10	`Done`
6	10	`60 % (on track)`
2	10	`20 % (behind)`

Boxes needed, with singular / plural: functions in conditions and results

```
ceil({qty} / {per_box}) <= 1 => 1 box
else => ceil({qty} / {per_box}) boxes
```

qty	per_box	Result
5	10	`1 box`
23	10	`3 boxes`

No `else` line: the field stays empty when nothing matches

```
{qty} >= 10 => Bulk order
```

qty	Result
12	`Bulk order`
3	*(empty)*

Giving a text result a numeric value — `[N]`

A text result such as `low` normally has the numeric value `0`. Put a number in square brackets in front of it, and the field shows the text but counts as that number in chained fields, conditions, and the Summary:

```
{probability} * {severity} <= 8 => [1] low
{probability} * {severity} <= 14 => [2] medium
else => [3] high
```

probability	severity	Displayed	Numeric value
2	4	`low`	`1`

3	3	`medium`	`2`
5	5	`high`	`3`

With three rows rated `low`, `high`, `medium` (1, 3, 2), the Summary can report on them:

Summary template	Output
`Highest risk level: {max risk_level}`	`Highest risk level: 3`
`Average: {round(avg risk_level; 1)}`	`Average: 2`

`[N]` works in every text result: the **If true** / **If false** branches of conditional mode (`[1] yes`, `[0] no`) and every multi-branch result. It applies to plain text only; a result with a calculation already has its own value.

Keeping the field empty for untouched rows

Empty fields count as `0`, so a new, untouched row matches the first rule of the risk matrix (`0 <= 8 => low`). To keep the field blank until the user makes a choice, add a rule with an empty result at the top:

```
{probability} * {severity} == 0 =>
{probability} * {severity} <= 8 => low
{probability} * {severity} <= 14 => medium
else => high
```

probability	severity	Result
(empty)	*(empty)*	*(empty)*
1	1	`low`

If Branches is empty

Because every formula property is always visible, it is easy to type the rules into the wrong box. In multi-branch mode, the plugin looks for rules in this order:

1. **Branches**, if it has at least one valid `condition => result` line.
2. **Condition (conditional mode)**, if *that* field has valid rule lines.
3. **Formula (simple mode)**, used as a single `else =>` rule, so the simple formula keeps working.
4. Otherwise the field stays empty.

The form editor preview shows a yellow ⚠ badge whenever cases 2–4 apply:

Badge	Meaning	What to do
⚠ Rules read from the Condition	The rules work, but are	Move them into **Branches**

field — move them to Branches	stored in the wrong property.	and clear **Condition**.
⚠ Branches empty — using the simple Formula	No rules anywhere; the simple formula is used.	Add rules to **Branches**, or switch back to **Simple** mode.
⚠ No branches defined	Nothing to evaluate — the field stays empty.	Add rules to **Branches**.

With valid rules in **Branches**, the preview lists them as `if … → …`, `else if … → …`, `else …`.

> **Note:** For security, formulas are evaluated by the plugin's own parser, never as JavaScript or PHP code. Arithmetic expressions may contain numbers, `+ - * / ( ) . ;` and the six listed function names; conditions may additionally contain the comparison operators, `&&` / `||`, and quoted texts.

Text mode — joining field values

When **Calculation mode** is set to **Text**, the field joins the values of other fields into a piece of text, **without any calculation**. Write the template in **Text template (text mode)**. Each `{fieldname}` is replaced by the field's value, and everything else stays exactly as written, including numbers and operators.

Text template	Row values	Result
`{title} {first} {last}`	title = Dr., first = Anna, last = Kovács	`Dr. Anna Kovács`
`{title} {first} {last}`	title = *(empty)*, first = Anna, last = Kovács	`Anna Kovács`
`{last} {first}`	first = Anna, last = Kovács	`Kovács Anna`
`Extras: {extras}`	extras = ☑ wifi ☑ breakfast	`Extras: wifi, breakfast`
`{status} / {country}`	status = paid, country = ◉ HU	`paid / HU`
`{qty} {unit}`	qty = 12, unit = pcs	`12 pcs`
`Risk: {risk_level}`	risk_level (Computed Field) shows `high`	`Risk: high`

- Dropdown, Radio, and Checkbox fields insert the option **value**; several checked values are joined with a comma.
- A Computed Field inserts its displayed text.
- Empty fields insert nothing, and the double spaces they would leave are removed (second row

above).
- There is no calculation: `{qty} * 2` gives `12 * 2`, not `24`. For a calculation, use a Simple-mode field and reference it from the text template.

Numeric value: if the result is a number (e.g. just `{qty}` → `12`), the field's numeric value is that number; otherwise it is `0`. `[N]` works here as well: `[1] {first}` shows `Anna` and counts as `1`.

If **Text template** is empty, the **Formula (simple mode)** text is used as the template, and the form editor preview shows a yellow ⚠ badge.

Switching between modes

Switching between modes does not delete anything. The formula, condition, branch, and rule texts all stay stored and are used again if you switch back. Only the active mode's properties are evaluated (except for the multi-branch fallback described above).

Chained computed fields

A Computed Field may reference another Computed Field in the same row. When the source field updates, it automatically triggers recalculation of any fields that depend on it. This works in every calculation mode and needs no configuration.

Example — net and gross line total

Field	Mode	Formula	Result (qty = 3, price = 100)
`line_net`	Simple	`{qty} * {price} | 2 Ft`	`300.00 Ft`
`line_gross`	Simple	`{line_net} * 1.27 | 2 Ft`	`381.00 Ft`

`line_gross` reads the **numeric value** of `line_net` (`300`), not its display text (`300.00 Ft`).

A text result such as `low` or `N/A` has the numeric value `0` in formulas, conditions, and Summary totals, unless it is given one with `[N]` (e.g. `[1] low`). A fixed text containing a number (e.g. `1990 Ft`) keeps its number (`1990`). If you need the score itself as well as a label, use two fields:

Field	Mode	Formula / Branches	Result (probability = 3, severity = 4)
`risk_score`	Simple (Hidden = YES if not shown)	`{probability} * {severity}`	`12`

`risk_level`	Multi-branch	`{risk_score} <= 8 => low`
		`{risk_score} <= 14 => medium`
		`else => high`
			`medium`

`risk_score` can then be totalled or averaged in the Summary template, while `risk_level` gives the readable label.

Display as — text or badge

By default the result appears in a (read-only) text field. For totals, labels, and levels, a field that cannot be typed into can be confusing, so **Display as** offers two other looks:

Display as	Looks like	Typical use
Input field *(default)*	a text field, as before	values the user may edit (with Read only = NO)
Plain text	bold text, no field border	line totals, full names
Badge	a small rounded label	levels and statuses: `low` / `medium` / `high`, `Paid` / `Unpaid`

Saving, chaining, the Summary template, email placeholders, and Advanced Conditional Logic work exactly the same with every option. Only the look on the form changes.

Colouring by value

With **Plain text** and **Badge**, the element also carries the result in a `data-rg-value` attribute, so it can be coloured with a few lines of CSS in your template, without any conditional logic plugin:

```
.rg-computed-display[data-rg-value="low"]    { --rg-badge-bg: #198754; --rg-badge-
color: #fff; }
.rg-computed-display[data-rg-value="medium"] { --rg-badge-bg: #ffc107; --rg-badge-
color: #212529; }
.rg-computed-display[data-rg-value="high"]   { --rg-badge-bg: #dc3545; --rg-badge-
color: #fff; }
```

With the risk matrix rules (`[1] low` / `[2] medium` / `[3] high`) and **Display as = Badge**, each row shows a green, yellow, or red label, updated live and independently per row, including duplicated rows. The attribute holds the displayed text (`high`, not `3`), so a result with a suffix is matched in full, e.g. `data-rg-value="1490 Ft"`.

- An empty result hides the badge (the element gets the `rg-computed-display--empty` class).
- The element is an `<output>` with `aria-live="polite"`, so screen readers announce changes.
- The general look is set by CSS variables; see CSS custom properties.

Computed fields in the Summary template

`{sum fieldname}`, `{avg fieldname}`, etc. also work when `fieldname` is a Computed Field. The Summary reads the numeric value, not the display text, so prefix/suffix formatting does not affect the totals.

Example: three rows whose `line_gross` fields show `381.00 Ft`, `127.00 Ft` and `254.00 Ft`:

Summary template	Output
`Total: {sum line_gross | 2} Ft`	`Total: 762.00 Ft`
`Average per line: {avg line_gross | 2} Ft`	`Average per line: 254.00 Ft`
`Lines: {count}`	`Lines: 3`

Row Duplication and Reordering

Duplicate button

When **Show duplicate button** is enabled on the Start marker, each row shows a **duplicate** button. Clicking it clones the row immediately below the original, copying all current values — including text inputs, select boxes, checkboxes, and Computed Field results. The cloned row is fully independent: changing values in the original does not affect the copy.

Drag-and-drop reordering

When **Sortable rows** is enabled on the Start marker, each row header displays a drag handle (⠿). Users can drag rows into any order. The row numbering, Summary template totals, and Computed Field chain all update instantly after a reorder.

> Both features respect the configured **Minimum rows** and **Maximum rows** limits.

Exporting Submissions with RG Export

When viewing submissions in the RSForm! Pro backend, an **RG Export** button appears in the toolbar next to the standard Export button. Clicking it opens a modal where you can choose:

Which rows to export:

- All rows
- Selected rows only (enabled when at least one submission is checked)

Export format:

Format	Description
CSV — pipe-separated	One row per submission. Group field values are concatenated with `|` separators.

CSV — expanded	One row per group repetition. Non-group fields repeat in every row. A `RepeatIndex` column indicates the repetition number.
XLSX — pipe-separated	Same as CSV pipe-separated, in native Excel format with a bold header row.
XLSX — expanded	Same as CSV expanded, in native Excel format.
XLSX — mixed (horizontal table)	Follows the form's field order. Plain fields appear as a header row + data row. Each repeatable group appears as a titled sub-table where each row is one repetition.
XLSX — mixed (field table)	Same mixed layout, but the repeatable group sub-table is transposed: one row per field, one column per repetition.
XLSX — mixed (cards)	Same mixed layout, but each repetition appears as its own labelled block (field: value pairs).

The three **mixed XLSX** formats are the most information-rich: they preserve the full structure of each submission, embedding the repeatable group data exactly where it appears in the form, with plain fields rendered in the same familiar header-plus-value style as standard spreadsheet exports.

> **Tip:** For submissions where the number of repetitions per group varies widely across submissions, the expanded formats (CSV or XLSX) are usually the easiest to process programmatically, since every repetition is already on its own row.

Visual Customisation with CSS

The plugin outputs a predictable set of CSS classes and data attributes. No template files need to be modified.

HTML structure

```
<!-- Header row - only present when "Show field labels" = HEADER_ROW -->
<div class="rg-header-row [rg-layout-horizontal] [rg-layout-styled] rg-header-row-
active">
  <div class="rg-header-cell">Caption 1</div>
  <div class="rg-header-cell">Caption 2</div>
</div>

<div class="rg-row [rg-layout-horizontal] [rg-layout-styled]"
     data-rg-row-index="0"
     data-rg-field-labels="all_rows|header_row|first_row_only|hidden">
  <div class="rg-row-header">
    <span class="rg-row-number">Row 1</span>
    <div class="rg-row-actions">
      <button class="rg-btn rg-btn-add-row">+</button>  <!-- EACH_ROW only -->
      <button class="rg-btn rg-btn-remove">…</button>
    </div>
  </div>
  <div class="rg-row-content">…</div>
</div>

<div class="rg-controls">
  <button class="rg-btn rg-btn-add">+ Add row</button>
</div>

<div class="rg-summary">…</div>  <!-- only if Summary template is set -->
```

CSS custom properties (variables)

As of version 1.4.7, the stylesheet defines its colours, spacing, and sizes as **CSS custom properties** on `:root`. To customise the appearance, redeclare the relevant variables in your own template CSS:

```
:root {
  --rg-accent: #2563eb;
  --rg-radius: 4px;
  --rg-styled-bg: #f5f5ff;
}
```

Variable	Default	Affects
`--rg-border-color`	`#dde0e5`	Row border colour.
`--rg-radius`	`6px`	Row border radius.
`--rg-row-bg`	`#fff`	Row background colour.
`--rg-row-padding`	`0.75rem`	Row inner padding.
`--rg-row-gap`	`0.75rem`	Vertical spacing between rows.

`--rg-row-hover-shadow`	`0 2px 8px rgba(0,0,0,0.08)`	Box-shadow on row hover.
`--rg-header-border-color`	`#eee`	Border under the row header.
`--rg-row-number-color`	`#555`	Text colour of the row number label.
`--rg-accent`	`#0fa564`	Add/+ button colour, accent border, focus ring base.
`--rg-accent-hover`	`#0d9058`	Add/+ button colour on hover.
`--rg-danger`	`#c0392b`	Remove (×) button colour.
`--rg-h-gap`	`0.75rem`	Gap between fields in HORIZONTAL rows.
`--rg-h-min-width`	`100px`	Minimum width of each field in HORIZONTAL rows.
`--rg-styled-bg`	`#f8fafc`	HORIZONTAL_STYLED row background colour.
`--rg-styled-hover-bg`	`#fff`	HORIZONTAL_STYLED row background on hover.
`--rg-styled-radius`	`8px`	HORIZONTAL_STYLED row border radius.
`--rg-styled-padding`	`0.85rem 1rem`	HORIZONTAL_STYLED row inner padding.
`--rg-styled-shadow`	`0 1px 4px rgba(15,30,60,0.08)`	HORIZONTAL_STYLED box-shadow.
`--rg-styled-shadow-hover`	`0 3px 10px rgba(15,30,60,0.14)`	HORIZONTAL_STYLED box-shadow on hover.
`--rg-label-color`	`#5a6577`	Field label colour in HORIZONTAL_STYLED rows.
`--rg-input-border`	`#d6dce5`	Input border colour in HORIZONTAL_STYLED rows.
`--rg-input-radius`	`6px`	Input border radius in HORIZONTAL_STYLED rows.
`--rg-focus-ri`	`rgba(15,165,100,`	Focus ring colour for inputs in HORIZONTAL_STYLED

`ng`	`0.15)`	rows.
`--rg-remove-bg-soft`	`#f1d3d0`	Circular × button background in HORIZONTAL_STYLED rows.
`--rg-remove-color-soft`	`#a3322a`	Circular × button icon colour in HORIZONTAL_STYLED rows.
`--rg-add-bg-soft`	`#d8f3e6`	Circular + button background in HORIZONTAL_STYLED rows.
`--rg-computed-bg`	`#f5f7fa`	Background colour of a read-only Computed Field input.
`--rg-computed-color`	`#3a4150`	Text colour of a read-only Computed Field input.
`--rg-drag-handle-color`	`#b0b8c4`	Colour of the drag handle icon when Sortable rows is enabled.
`--rg-display-color`	`#1f2937`	Text colour of a Computed Field shown as **Plain text**.
`--rg-display-weight`	`600`	Font weight of a Computed Field shown as Plain text or Badge.
`--rg-badge-bg`	`#e8eaf0`	Background of a Computed Field shown as **Badge**. Can be set per value, see Colouring by value.
`--rg-badge-color`	`#1f2937`	Text colour of a Badge.
`--rg-badge-radius`	`999px`	Corner radius of a Badge.
`--rg-badge-padding`	`0.2em 0.75em`	Inner padding of a Badge.
`--rg-badge-font-size`	`0.875em`	Font size of a Badge.

Where the plugin's CSS and JavaScript are loaded

Since version 1.7.15 the plugin adds its stylesheet and scripts **only to pages that actually show a form with a Repeatable Group or a Computed Field**. Pages without such a form (your home page, articles, shop pages) no longer load them. This works in every place a form can appear: the RSForm! Pro menu item, the RSForm! Pro module, a form embedded with `{rsform N}` (also when the System - RSForm! Pro plugin inserts it at the end of page rendering), a popup that opens the form in its own page

(`tmpl=component`), multi-page forms, and the form shown again after a failed submission. The **Submissions – View** menu item does not need the files: its tables, thumbnails and totals are styled inline.

Things to keep in mind if you customise your site:

- **Your own CSS or JavaScript** that targets `.rg-*` classes, listens to `rg:*` events or uses `window.RepeatableGroupField` will only find them on pages with a group. Put such code in the form's own CSS/JS settings, or check first: `if (window.RepeatableGroupField) { … }`.
- **Forms inserted late in page rendering** (for example by a third-party plugin that places content after Joomla has built the page head) are detected automatically: the plugin adds the missing files just before `</body>`. You can recognise these tags by the `data-rg-fallback="1"` attribute.
- **Page optimisers** that combine or rename JavaScript and CSS files: exclude the files under `media/plg_system_repeatablegroup/` from combining. Otherwise the plugin cannot see that the files are already on the page and may add them a second time.
- **Popups that insert the form's HTML into an already loaded page** with JavaScript are not supported, because the group is set up when the page loads. Use a popup that loads the form in its own frame instead.

File Uploads Inside a Group

Since version 1.7.0 the standard RSForm! Pro **File Upload** field can be placed inside a repeatable group. Each row gets its own upload input, and the files are stored per row.

Setting it up

Add a File Upload field between the Start and End markers and configure it exactly as you would anywhere else in the form: destination folder, accepted file extensions, maximum file size, file name prefix, image thumbnails, email attachments. No extra settings are needed.

All three upload modes work per row:

Mode	How it behaves in a row
Single file	One file per row.
Multiple	One input with multiple file selection per row.
Multiple + **Add more files** button	The "Add more" button adds extra inputs inside that row only.

Minimum / maximum files are counted per row. With *Maximum files* = *3*, every row can hold up to three files. **Required** means every row must contain at least one file.

Validation

Every file is checked on the server with the field's own rules: allowed extensions, image check, maximum

size, and minimum/maximum number of files. Error messages name the field and the row, e.g. *Documents (row 2): The file extension is not allowed.*

For security, script and executable extensions (for example `.php`, `.phtml`, `.phar`, `.exe`, `.sh`) are always rejected inside groups, even if the field has no accepted-extensions list.

Server limits

PHP limits how many files and how much data one submission can contain (`max_file_uploads`, `post_max_size`, `upload_max_filesize`). A form with many rows and several files per row reaches these limits quickly — the default `max_file_uploads` is only 20 files per submission. The plugin checks the selected files before the form is sent and shows a clear message if a limit would be exceeded. If your forms need more, ask your host to raise these values.

Validation rules inside a group

(1.8.3) The **Validation rule** of a Text Box or Textarea field inside a group (numeric, e-mail, regex, your custom rules in `customvalidation.php`, *Multiple rules*) is checked on every row when the form is submitted. Empty cells are not checked unless the field is **Required**. The message names the field and the row and uses the field's *Validation message* — for example *Qty (row 2): Qty must be a whole number.* If the message is empty, *Invalid value.* is shown.

Three rules are skipped inside groups because they compare a single value with something outside the row: *Same as*, *Unique field* and *Unique field per user*.

After a failed submission

Browsers never allow a web page to re-fill a file input. If the form is rejected on the server (for example because of a missing required field), the other field values are restored, but the files must be selected again. A notice tells the user to do so.

Emails, PDFs, and submissions

- **Placeholders:** `{fieldname}`, `{fieldname:value}` and `{fieldname:list}` output download links per row. The group tables (`{groupId:table}`, `{groupId:cards}`, etc.) show the files of each row as links. `{fieldname:filename}`, `{fieldname:path}`, `{fieldname:localpath}`, `{fieldname:image}` and `{fieldname:count}` work as in RSForm! Pro and cover the files of all rows.
- **Images (1.7.8):** `{fieldname:thumb}` shows uploaded image files as images, row by row. Set **Show uploaded images as images** on the End marker to display images in the group tables and cards as well. See Images from File Upload fields.
- **Email attachments:** if the field's *Attach to email* option is set, the files of every row are attached.
- **Submissions list and editor:** the files are listed per row (`#1: a.pdf, b.jpg`) with download links. In the submission editor this column is read-only.
- **Deleting submissions:** deleting a submission (or selected submissions) also deletes its uploaded files.

> **Note:** RSForm! Pro's "Delete all submissions" function does not notify plugins, so files uploaded inside groups stay on the server in that case. Delete submissions individually or in a selection if you want the files removed as well.

Email and PDF Placeholders

The placeholders below work everywhere RSForm! Pro resolves placeholders: user and admin emails, the Thank You message, PDFs, and the frontend **View Submissions** menu item (see Frontend "View Submissions" pages).

Single-field placeholders

Placeholder	Output
`{fieldname:value}` (or `{fieldname}`)	All values from that field, separated by pipes: `val1 | val2 | val3`
`{fieldname:list}`	Values separated by line breaks (` `)
`{fieldname:table}`	A single-column HTML table showing values for that field only, with a row-number column.
`{fieldname:first}`	The value of the first row. *(1.7.8)*
`{fieldname:last}`	The value of the last row. *(1.7.8)*
`{fieldname:inline}`	The non-empty values, separated by commas: `Apple, Pear`. *(1.7.8)*
`{fieldname:count}`	The number of non-empty values. For a File Upload field: the number of uploaded files. *(1.7.8)*

Group placeholders

`groupId` is the **Group ID** configured on the Start marker (or the Start field's Field Name, if Group ID is left empty).

Placeholder	Output
`{groupId:table}`	Transposed table: one row per submitted entry, one column per field, with a leading "#" row-number column and a header row of field captions.
`{groupId:table_noheader}`	Same as `table`, without the header row of field captions.

`{groupId:table_nonum}`	Same as `table`, without the leading "#" row-number column.
`{groupId:table_plain}`	Same data, with neither header row nor row-number column.
`{groupId:table_fields}`	**Field-major** view: one row per field, one column per submitted entry.
`{groupId:cards}`	**Card view**: each submitted entry becomes its own small table with a numbered header and one *field label : value* row per field.
`{groupId:summary}`	Renders the group's **Summary template** with all `{...}` expressions replaced by their computed values. Empty if no Summary template is configured.
`{groupId:count}`	The number of submitted rows. `0` if the group was left empty. *(1.7.8)*
`{groupId:inline}`	A one-line summary: the non-empty values of each row joined with " – ", rows separated by "; ". Example: `Apple – 2; Pear – 5`. File fields show file names. *(1.7.8)*

If a submission has no data for a group, for example a submission made before the group was added to the form, every group and field placeholder resolves to an empty string (`{groupId:count}` and `{fieldname:count}` to `0`). The placeholder text never appears in the output.

"Raw" placeholders for PDF generation

Every table and card placeholder above also has a `_raw` variant: `{groupId:table_raw}`, `{groupId:table_noheader_raw}`, `{groupId:table_nonum_raw}`, `{groupId:table_plain_raw}`, `{groupId:table_fields_raw}`, `{groupId:cards_raw}`.

These output the **same table structure without any inline `style="..."` attributes**. Instead, semantic CSS classes are added (`rg-pdf-table`, `rg-pdf-th`, `rg-pdf-td`, `rg-pdf-num`, `rg-pdf-card-title`, and `rg-pdf-img` for images).

> **Email vs. PDF:** most email clients strip `<style>` blocks, so the regular placeholders (with inline styles) are recommended for email bodies. PDF generators (e.g. DOMPDF, mPDF) support CSS classes correctly, so the `_raw` placeholders with a stylesheet in your PDF template are recommended for PDF attachments.

Images from File Upload fields

(1.7.8) Image files uploaded in a File Upload field inside a group can be shown as images instead of download links. This applies to jpg, jpeg, png, gif and webp files. Other file types, including SVG, always

stay download links.

Per placeholder:

Placeholder	Output
`{fieldname:thumb}`	Row by row: each image file as an image (clicking it downloads the file), each other file as a link. Rows are separated by line breaks.
`{fieldname:thumb_raw}`	The same for PDFs: the image uses the local file path, has `class="rg-pdf-img"` and a `width` attribute, and has no inline style.

These two placeholders always show images, whatever the End marker setting is.

In the tables and cards: on the **End marker**, set:

Property	Effect
Show uploaded images as images (placeholders)	**Yes**: the group tables and cards (`{groupId:table…}` , `{groupId:cards…}` , `{fieldname:table}`), `{fieldname:first}` and `{fieldname:last}` show image files as images. Default: **No** (links, as before).
Image max width (px)	Maximum image width, 16–2000 pixels. Default: 120. In `_raw` variants this is the image width.

`{fieldname}` , `{fieldname:value}` , `{fieldname:list}` and `{fieldname:inline}` always output links.

Where the image comes from:

- **Emails and web pages:** the image is served by the plugin through the same secured link RSForm! Pro uses for file downloads, so it also works when the upload folder is not publicly accessible. The plugin checks that the file really is an image before serving it.
- **PDFs:** `_raw` placeholders, and every placeholder when the PDF of a View Submissions page is generated, use the local file path, because the PDF generator cannot download files through a link.

> **Tip:** some email clients block images until the reader allows them. The image is a link to the file, so the file stays available either way.

Frontend "View Submissions" pages

(1.7.8) RSForm! Pro's **Submissions - View** menu item (*View Submissions*) has three layout fields. The plugin's placeholders work in two of them:

Layout field	Placeholders	Recommended

HTML Layout	Only `{formdata}` (RSForm! Pro's own)	—
Row Layout (one listing row per submission)	All	Compact ones: `{groupId:count}`, `{groupId:inline}`, `{fieldname:first}`, `{fieldname:inline}`, `{fieldname:thumb}`
Details Layout (the detail page and its PDF)	All	Full tables: `{groupId:table}`, `{groupId:cards}`, `{groupId:summary}`. In the PDF part, use the `_raw` variants.

On these pages the plugin **builds only the placeholders that the layout actually contains**, so long listings stay fast. Values from `{fieldname}` and `{fieldname:value}` are HTML-escaped, because the page may be public.

Example Row Layout (hides the group column when the group is empty):

```
<tr>
  <td>{details}{FullName:value}{/details}</td>
  <td>{if {items:count}}{items:count} items: {items:inline}{/if}</td>
  <td>{photos:thumb}</td>
</tr>
```

Example Details Layout:

```
<h3>{FullName:value}</h3>
{items:table}
{items:summary}
{detailspdf}Download PDF{/detailspdf}
```

Submissions Directory

(1.8.0) RSForm! Pro's **Submissions Directory** is the frontend listing of a form's submissions, with a details page, PDF, CSV download and — for the user groups you allow — editing. From version 1.8.0 repeatable groups are fully supported there: no more raw JSON in the listing, the details page, the CSV file or the edit form.

How a group appears in the Directory field list

In the form's Submissions Directory settings every field has a row with the *Listing*, *Searchable*, *Editable*, *In details* and *In CSV* checkboxes. For a repeatable group there are two kinds of rows:

Row in the field list	Listing	In details	In CSV	Editable

A field inside the group (for example `qty`)	Its own column: one line per group row, so several group columns line up next to each other	`{qty:value}` → `10 | 5 | 2`	`10 | 5 | 2`	*No checkbox* — see Editing a group on the frontend
The Start marker (the whole group)	One summary column: each group row on its own line, cells joined with `–`	The whole group as a table (`{groupId:table}`)	A one-line summary: `Bolt – 10 – 100; Nut – 5 – 25`	The grid editor

The **End marker is not listed** — it has nothing to show.

You can combine the two freely: tick *Listing* on `product` and `qty` to get two aligned columns, or tick it on the Start marker to get one compact column for the whole group.

Listing

- Empty cells keep their line, so row 2 of `product` always sits next to row 2 of `qty`.
- A **File Upload** field inside the group shows the download links of each row on its own line.
- **Searchable** works for fields inside the group: the search matches any text in any row. The Start marker's summary column cannot be searched.
- **Sorting** by a group column and the *is* / *is not* dynamic filters compare the whole stored list, not single rows, so they are not useful on group fields. Use *contains* filters instead.

Details page and PDF

With **Auto-generate layout** switched on, ticking *In details* on the Start marker adds the whole group as a table to the details page. Open and save the Directory settings once after updating to 1.8.0 so the layout is regenerated.

If you write the details layout by hand, every placeholder from Email and PDF Placeholders works in it, for example:

```
<h3>{FullName:value}</h3>
{items:table}
<p>{items:summary}</p>
```

As on the *View Submissions* pages, the plugin builds only the placeholders the layout (and the PDF file name) actually contains, and `{fieldname}` / `{fieldname:value}` values are HTML-escaped. For the PDF, prefer the `_raw` variants. Directory **emails** receive every placeholder.

CSV download

Column	Content

A field inside the group	The rows joined with `|`: `Bolt | Nut | Washer`
A File Upload field inside the group	The download URLs: files of one row separated with `,`, rows with `|`
The Start marker	A one-line summary of the group: cells joined with `–`, rows with `;`

For row-per-line exports use **RG Export** in the backend (see Exporting Submissions with RG Export).

Editing a group on the frontend

Tick **Editable** on the group's **Start marker**. Users who are allowed to edit a submission then get a grid on the edit page: one column per field of the group, one line per row.

Field inside the group	Cell in the grid
Text Box, Calendar, Hidden and other single-value fields	Text input
Textarea	Multi-line text area
Radio Group, Dropdown (single choice)	Dropdown with the field's items. A stored value that is no longer among the items is kept as an extra option.
Checkbox Group, Dropdown with *Multiple*	Text input; several values are separated with `,`
Computed Field	*(1.8.2)* Read-only box. It is **recalculated in the browser** whenever another cell of the same row changes, with the same formula as on the form. When the page opens, the stored values are shown unchanged.
File Upload	*(1.8.2)* The row's files as download links, each with a **Delete** checkbox, and a file chooser for new files. In a single-file field a new file **replaces** the old one; with *Multiple* the new files are added to the kept ones.

- **Add row / remove row** follow the Start marker's *Minimum rows* and *Maximum rows*; the buttons use its button texts.
- **Removing a row also deletes the files** uploaded in that row.
- On save the plugin checks the row limits, the **Required** fields of every row and *(1.8.2)* the fields' **validation rules**. If something is wrong, **nothing is saved** (not even the other fields of the submission, and no file is uploaded or deleted), the messages name the group, the field and the row — for example *Order items: E-mail (row 2): Please enter a valid e-mail address* — and the edit form

comes back with the values the user typed and the files ticked for deletion. Files chosen for upload have to be chosen again.

- *(1.8.2)* **Validation rules** of Text Box and Textarea fields run on every non-empty cell, with RSForm! Pro's own rules (including your custom rules in `customvalidation.php`) and *Multiple rules*. The message is the field's *Validation message*; if that is empty, *Invalid value.* Three rules are skipped in the grid because they cannot work on a single cell: *Same as*, *Unique field* and *Unique field per user*.
- *(1.8.2)* **File Upload rules** apply per row, as on the form: accepted extensions, maximum size, images only; with *Multiple*, *Minimum files* / *Maximum files* count the kept and the new files together. A **required** file field needs at least one file in every row — a row that has no file blocks the save until one is uploaded. New files are stored by RSForm! Pro's own upload code (destination folder, prefix, thumbnails), only after every check has passed.
- A submission made **before the group existed** (its fields hold plain values, not rows) shows an empty grid. Saving it without adding a row leaves those old values untouched.
- If the Directory has a custom **Save script** that rejects the save after the plugin ran, the changes to the group are already stored. Put group-related checks in the form's own validation instead.

Why fields inside a group have no *Editable* checkbox. RSForm! Pro would edit such a field with its normal input — a single text box, a set of radio buttons — although the stored value is a list with one entry per row. Saving that form empties Radio / Checkbox / Dropdown fields, and a validation rule on the field blocks the save of the whole submission. The plugin therefore removes the checkbox, and edits the group as one unit through the Start marker. If a field inside a group was marked *Editable* before the update, the plugin clears that mark the first time the edit page is opened.

Styling the grid. The editor is wrapped in `.rg-dir-edit`; rows are `tr.rg-dir-row`, cells `td.rg-dir-cell` (plus `rg-dir-cell--text`, `--textarea`, `--select`, `--computed`, `--file`), inputs `.rg-dir-input`, the buttons `.rg-dir-add` and `.rg-dir-remove`. In file cells: `.rg-dir-files`, `.rg-dir-file-list`, `.rg-dir-file`, `.rg-dir-file-del` (checkbox) and `.rg-dir-file-input`. Below 640 px every row becomes a card with the field caption above each cell. Colours and spacing can be changed with CSS variables:

```
.rg-dir-edit {
   --rg-dir-border: #cfd8dc;
   --rg-dir-head-bg: #eceff1;
   --rg-dir-cell-min-width: 140px;
}
```

Available variables: `--rg-dir-border`, `--rg-dir-head-bg`, `--rg-dir-head-color`, `--rg-dir-num-color`, `--rg-dir-note-color`, `--rg-dir-cell-padding`, `--rg-dir-cell-min-width`, `--rg-dir-radius`.

Viewing and Exporting Submissions

Viewing submissions

Go to **Forms** → **your form** → **Submissions** in the RSForm! Pro backend. Submitted group data is stored as JSON arrays. In the submissions editor, the plugin replaces the raw JSON display with an editable grid UI: each row of the group appears as a row in a compact table, with individual cells you can edit directly. The JSON is updated automatically when you save the submission. File Upload fields inside a group are shown as a read-only list of download links per row (`#1: a.pdf, b.jpg`), both in the submissions list and in the editor.

Exporting submissions

The submissions list toolbar shows two export options:

- **Export** (standard RSForm! Pro button) — exports all fields to CSV, XLSX, ODS, or XML. Group field values appear as pipe-separated strings in a single cell.
- **RG Export** (added by this plugin) — opens a dedicated export modal with seven formats optimised for repeatable group data. See Exporting Submissions with RG Export above.

Field Compatibility

✅ Full support

Field type
Text
Textarea
Email
Number
Password
Hidden
Dropdown (Select)
Radio
Checkbox
Date
Calendar (RSForm! Pro built-in)
jQuery Calendar (RSForm! Pro built-in)
Repeatable Group: Computed Field

File Upload (single, Multiple, Add more files) — see File Uploads Inside a Group

> **Calendar fields:** the plugin automatically converts the RSForm! Pro YUI/jQuery calendar widget to a native HTML5 `<input type="date">` within each row. This ensures the date picker works correctly in every row — including cloned and duplicated rows — without any additional configuration. The field's **Date format** property should be set to `Y-m-d` to match the native date input format. The conversion is applied silently; the rest of the field's behaviour (name, validation, stored value) is unchanged.

⚠ Partial support

Field type	Notes
CAPTCHA	Will only function in the first row. Do not place a CAPTCHA inside a repeatable group.

❌ Not supported

Field type	Reason
Repeatable Group: Start / End	Nested groups are not supported.
Timeslot Field	Maintains internal availability state that does not work with row cloning.
Form Expiry Field	Form-level field — only one instance per form makes sense.

> The form editor displays a warning badge next to any incompatible field placed inside a repeatable group.

Advanced Conditional Logic (ACL plugin)

The **Advanced Conditional Logic** plugin (sold separately) is fully compatible with Repeatable Group fields as of ACL v1.3.53. Each row evaluates its own conditional rules independently: for example, checking a checkbox in row 2 shows/hides a field in row 2 only, without affecting row 1 or row 3. Rules also apply automatically to newly added, duplicated, and session-restored rows.

> Requires **Advanced Conditional Logic v1.3.53** or later.

JavaScript Events for Developers

Since version 1.7.10, the group sends **browser events** when something happens in it: a row is added, duplicated, removed or moved, a computed value changes, or the row limit is reached. Your own script can react to them, for example to ask for confirmation before a row is deleted, show a message at the maximum number of rows, or send data to an analytics tool.

This is optional and needs a little JavaScript. The plugin itself works the same without it.

Listening to an event

Every event starts on the group's **Start** field block and bubbles up, so you can listen on the form or on the whole page:

```
document.addEventListener('rg:rowadded', function (e) {
   console.log('New row in', e.detail.groupId, 'at position', e.detail.index + 1);
});
```

Add the script to the form in RSForm! Pro under **Properties** → **CSS and JavaScript**, or in your template.

`e.detail` always contains:

Property	Meaning
`groupId`	The group's ID: the **Group ID** of the Start field, or its Field Name if Group ID is empty. Use it to tell groups apart when a form has more than one.
`count`	The number of rows after the change.
`group`	The group object (see The group object).

Events

Event	When	Extra `e.detail` properties
`rg:ready`	The group has been built on the page.	`rows` (the row elements), `atMin`, `atMax`
`rg:rowadded`	A row was added with an Add button.	`row`, `index`
`rg:rowduplicated`	A row was duplicated.	`row` (the copy), `index`, `sourceRow`, `sourceIndex`
`rg:beforeremove`	A row is about to be removed. **Can be cancelled** with `e.preventDefault()`.	`row`, `index`

`rg:rowremoved`	A row was removed.	`row` (no longer on the page, but its fields can still be read), `index` (its position before removal)
`rg:reordered`	A row was moved with drag and drop. Not sent if the row was dropped where it was.	`row`, `from`, `to`, `rows` (new order)
`rg:computed`	The result of a Computed Field changed.	`row`, `index`, `field` (field name), `element`, `value` (as shown), `numeric`, `previousValue`, `previousNumeric`
`rg:limit`	The group reached or left its minimum or maximum number of rows.	`limit` (`"min"` or `"max"`), `reached` (`true` / `false`)

Row positions (`index`, `from`, `to`, `sourceIndex`) start at **0**: the first row is `0`.

Order and timing:

- While the page loads, the group builds its starting rows (minimum rows, rows restored after a failed submission) **without** sending events. It then sends one `rg:ready` with the starting state.
- A new or duplicated row's first computed values do not send `rg:computed`. They are already filled in when `rg:rowadded` / `rg:rowduplicated` arrives.
- `rg:limit` follows the row event that caused it: first `rg:rowadded`, then `rg:limit`.
- `rg:computed` is sent only when the displayed text or the numeric value actually changes, not on every keystroke.
- With **Maximum rows** set to `0` (no limit), no `max` event is sent.

Examples

Ask before removing a row

```
document.addEventListener('rg:beforeremove', function (e) {
   if (!window.confirm('Remove row ' + (e.detail.index + 1) + '?')) {
      e.preventDefault();
   }
});
```

Show a message at the maximum number of rows

```
document.addEventListener('rg:limit', function (e) {
   if (e.detail.limit !== 'max') return;
   var note = document.getElementById('rg-max-note');
   if (note) note.hidden = !e.detail.reached;
});
```

Add the note to the form, e.g. in a **Free Text** field after the End field: `<p id="rg-max-note" hidden>You have reached the maximum number of items.</p>`

Highlight a row when a computed value is over a limit

```
document.addEventListener('rg:computed', function (e) {
   if (e.detail.field !== 'total') return;
   e.detail.row.classList.toggle('row-over-limit', e.detail.numeric > 100000);
});
```

Here `total` is the name of the Computed Field. For colouring by the **displayed** value, the CSS-only method in Colouring by value is usually simpler.

Only one group of several

```
document.addEventListener('rg:rowadded', function (e) {
   if (e.detail.groupId !== 'attendees') return;
   // …
});
```

The group object

`e.detail.group` is the group itself. You can also get it at any time with `window.RepeatableGroupField.get('groupId')`. This is useful if your script loads after the page has been built and missed `rg:ready`.

Member	What it does
`rows`	The current row elements, in order.
`addRow()`	Adds a row at the end and returns it. Returns nothing at the maximum.
`duplicateRow(row)`	Duplicates `row` below itself and returns the copy.
`removeRow(row)`	Removes `row` (sends `rg:beforeremove` first). Returns `true` if it was removed, `false` if not (minimum reached or cancelled).

`window.RepeatableGroupField.all()` returns every group on the page.

(1.8.2) `window.RepeatableGroupField.evalComputedRow(element)` recalculates every Computed Field inside `element` from the values of that element's fields — the Submissions Directory editor uses it for its rows. It sends no events.

```
// Add three rows from your own button
var g = window.RepeatableGroupField.get('items');
if (g) { g.addRow(); g.addRow(); g.addRow(); }
```

> The events report what happened; they do not replace the plugin's own settings. Use **Minimum rows** / **Maximum rows** for limits and the Advanced Conditional Logic plugin for showing or hiding fields.

Frequently Asked Questions

Can I have more than one repeatable group on the same form? Yes. Each group must have its own Start and End markers with a unique Field Name (and optionally a unique Group ID).

Can I nest one repeatable group inside another? No. Nested groups are not supported.

Is the data saved even if the user only fills in one row? Yes. A minimum of one row is always saved.

What happens if the user submits the form with empty rows? Empty rows are not saved. Only rows where at least one field contains a value are included in the submission.

Does conditional logic work inside a group? Native RSForm! Pro conditional logic works within a single row. With **Advanced Conditional Logic v1.3.53+**, full per-row conditional rules are supported — each row evaluates independently. Cross-row or cross-group conditions are not supported.

The group does not work, and a yellow warning appears under the group heading — why? The plugin shows this warning when it cannot find the group's End marker after the Start marker, or when the two are on different form pages. Check the order of the fields in the form editor. Every Start marker needs its own End marker, with the group's fields between them and on the same page.

I set "Show field labels" to HEADER_ROW, but nothing shows at the top of the group — why? The header row is only rendered if the group's fields have a **Caption** set. Make sure each field inside the group has a meaningful Caption.

I set "Add row button position" to EACH_ROW — where did the original Add row button go? It's still in the page (along with the hidden fields the plugin needs for submission), just visually hidden. The "+" button on each row inserts a new empty row immediately below that row.

The RG Export button is not visible in the submissions toolbar — why? Make sure the plugin is enabled and that Joomla's cache has been cleared after the latest update (System → Clear Cache). The button only appears on the submissions list view, not on individual submission edit screens.

Which export format should I use?

- If you want to process the data in Excel or a spreadsheet app: **XLSX — mixed** gives the most readable result, with plain fields and group tables laid out in natural form order.
- If you need to import the data into another system: **CSV expanded** or **XLSX expanded** gives one row per group repetition, which is easiest to process programmatically.
- If you just want a quick overview with all data in one cell: **CSV pipe-separated** or **XLSX pipe-separated**.

Which placeholder should I use for a PDF attachment? Use one of the `_raw` placeholders (e.g. `{groupId:table_plain_raw}` or `{groupId:cards_raw}`) together with a `<style>` block in your PDF template that targets the `rg-pdf-*` classes.

My Summary template shows nothing where I expect a number — why? Check that the field name inside `{...}` exactly matches the **Field Name** of a field inside the group (case-sensitive), and that the expression's syntax is valid. An expression that can't be computed renders as empty rather than showing an error.

Does `{sum ...}` work with Dropdown, Radio, and Checkbox fields? Yes — as long as the option **values** (not labels) are numbers. For Checkboxes, the checked values within each row are added together first, then summed across rows.

Does `{sum fieldname}` work when `fieldname` is a Computed Field? Yes. The Summary template reads the underlying numeric value stored in the Computed Field, not the display text. Prefix or suffix text in the formula (e.g. `"Amount: 300.00 Ft"`) does not affect aggregation.

My Computed Field formula uses parentheses but the field stays empty — why? Check that the `| N` rounding token (if used) comes **after** the closing parenthesis of the entire expression, not inside it. The parser splits on the last `|` character: everything before it is the arithmetic expression, everything after is the precision and optional suffix. Correct example: `{net} * (1 + {vat} / 100) | 2 €`. Before version 1.7.3, a formula that *started* with a parenthesis, a number, or a minus sign (e.g. `({a} + {b}) / 2`, `100 - {a}`) was also evaluated incorrectly; update to 1.7.3 or later.

Can a Computed Field reference another Computed Field? Yes — chaining is fully supported. When the upstream field recalculates, it automatically triggers recalculation of any downstream fields that reference it.

Can a Computed Field show one result when a condition is true and a different result when it is false? Yes — set **Calculation mode** to **Conditional**. Enter a comparison expression in the **Condition** field (e.g. `{qty} > 0`), then fill in the **If true** and **If false** formula fields. Both branches support the same prefix/suffix/rounding syntax as the simple formula. Only one branch is evaluated at a time, depending on whether the condition is currently met.

How do I turn a number into three or more categories (e.g. low / medium / high)? Use **Multi-branch** mode and write one `condition => result` rule per line in **Branches**, with an `else =>` line last. See Multi-branch mode for a complete risk-matrix example.

Which functions can I use, and why does `round({x}, 2)` not work? `round`, `floor`, `ceil`, `abs`, `min`, and `max` (since 1.7.4), in Computed Field formulas, conditions, multi-branch rules, and the Summary template. Arguments are separated by a **semicolon**, because the comma is the decimal separator: write `round({x}; 2)`. See Functions.

Can a label like "high" also count as a number? Yes (since 1.7.4): write `[3] high`. The field shows `high`, and chained fields, conditions, and the Summary use `3`. See Giving a text result a numeric value.

How do I check which option was chosen in a Dropdown or Radio field? Compare the field with the option **value** in quotes, e.g. `{status} == "paid"` (since 1.7.5). The label is not used, and letter case

does not matter. For Checkbox groups, `{extras} == "parking"` is true if *parking* is among the checked values. See Text comparison.

Can a Computed Field show a full name or other text built from several fields? Yes: set **Calculation mode** to **Text** (since 1.7.5) and write e.g. `{title} {first_name} {last_name}` into **Text template**. See Text mode.

Can a Computed Field look like a coloured label instead of an input field? Yes (since 1.7.6): set **Display as** to **Badge** (or **Plain text**) and colour it per value with CSS, e.g. `.rg-computed-display[data-rg-value="high"] { --rg-badge-bg: #dc3545; --rg-badge-color: #fff; }`. See Display as — text or badge.

Can I calculate inside a condition, or combine several conditions? Yes (since 1.7.1). Both sides of a comparison can be calculations, e.g. `{qty} * {price} >= 1000`, and comparisons can be joined with `&&` (and) and `||` (or), grouped with parentheses.

My multi-branch Computed Field stays empty — why? Check the field's preview in the form editor. A yellow ⚠ **No branches defined** badge means no rules were found: make sure the rules are in the **Branches (multi-branch mode)** property, one per line, each containing `=>`. If the badge says the rules were read from the Condition field, they work, but move them to **Branches** to keep the field tidy. If there is no badge, check that every `{fieldname}` matches a field in the same group (case-sensitive) and that an `else =>` line exists if every row must show a value.

Can Advanced Conditional Logic react to a text result such as "high"? Yes, but use a text rule (*equals* `high`), not a numeric one (*less than or equal*). A text result has no numeric value, so numeric comparisons do not match it. If you need numeric ranges, run the rules on a separate score field instead.

Can I use a comma as a decimal separator in conditions? Yes. Both `.` and `,` are accepted in condition literals and in field values used in conditions. The plugin normalises commas to points before evaluation.

Does drag-and-drop reordering affect the submitted data order? Yes. Rows are stored and submitted in the visual order at the time of submission. Reordering before submitting changes the order in emails, exports, and placeholders.

A user's files disappeared after the form showed an error — why? Browsers do not allow a page to re-fill file inputs, so after a server-side validation error the files must be selected again. The plugin shows a notice about this. All other row values are restored automatically.

Can I upload several files in one row? Yes. Enable **Multiple** on the File Upload field (optionally with the **Add more files** button). Minimum and maximum file counts apply to each row separately.

What languages are supported? The plugin interface is available in English and Hungarian. Additional languages can be added by translating the language files.

Support

For questions, customisation requests, or to report issues, contact:

extensions.blaho.hu — RSForm! Pro extensions and Joomla development.