Advanced Conditional Logic — RSForm! Pro Plugin

Documentation version: 1.3.61

Product Description

The Advanced Conditional Logic plugin provides a comprehensive solution for RSForm! Pro forms, enabling dynamic field visibility, value manipulation, and intelligent form logic based on fully configurable conditions. The plugin adds a new field type to RSForm! Pro: the Advanced Conditional Logic field, which evaluates rules in real-time and executes a series of actions on other form fields — showing or hiding content, setting values, adjusting requirements, modifying classes and inline styles, and manipulating option lists — without writing a single line of custom JavaScript. When custom JavaScript *is* needed, the `call_function` action calls your own function at the moment the condition changes state.

Perfect for registration forms, e-commerce checkouts, surveys, scholarship applications, or any form that needs to respond intelligently to user input, submission counts, time windows, or user identity. Administrators configure all logic through a visual builder interface directly in the form editor, with auto-save and instant feedback.

The plugin integrates seamlessly with RSForm! Pro's existing features, including all standard field types, validation rules, and multilingual support. Conditions are also evaluated server-side on submission — preventing manipulation and ensuring reliable behavior even when JavaScript is disabled.

Key Features

- **5 Condition Types** — Submission count, field value, time-based, user-based, and combined AND/OR rules
- **31 Action Types** — Show, hide, require, optional, enable, disable, set value, set default value (editable), calculate, clear value, set HTML, set/add/remove class (input), set/add/remove class (block), add/remove style (input), add/remove style (block), add/remove option, show/hide/enable/disable option, check/uncheck option, select option, call JavaScript function
- **Editable Default Values** — `set_default_value` fills a field when the condition switches, then leaves it to the visitor (v1.3.61)
- **Custom JavaScript Callbacks** — `call_function` calls a named global function only on state change (true ↔ false), with full context including the Repeatable Group row; no `eval()`
- **Inline Style Actions** — Add/remove CSS declarations on the input or its block wrapper; removal restores the original inline value
- **Field-type-aware Action List** — Only contextually valid actions appear for the selected target field type
- **Combined Conditions** — AND / OR logic grouping any mix of condition types into a single rule

- **Field Value Condition** — Value input auto-switches to a populated dropdown for select, radio, and checkbox fields
- **Checkbox-aware Logic** — Correct `contains` / `not_contains` operators for multi-value checkbox submissions
- **Operator Filtering** — Only valid operators shown per field type, eliminating configuration errors
- **Regex Conditions** — `matches_regex` and `not_matches_regex` operators for pattern-based text validation (postal codes, phone numbers, custom formats) with real-time feedback
- **Initial State Management** — Fields targeted by `hide` actions are hidden server-side before JavaScript runs, eliminating flash-of-content on page load without breaking flex/grid layouts
- **Lightweight Pages** — The plugin's CSS and JavaScript are loaded only on pages that actually display a form with conditional rules
- **Multilingual Support** — Option dropdowns show base values with translated labels, ensuring conditions work correctly across all languages
- **Time-Based Conditions** — Unix timestamp comparison with built-in datetime picker
- **User-Based Conditions** — Logged-in status, guest detection, user ID and Joomla user group
- **Visual Action Builder** — Ordered action lists, delete buttons, clear field and action labels
- **Real-time Admin Save** — All changes auto-save directly from the visual builder
- **Execution Order** — Configurable priority when multiple conditional fields coexist on one form
- **Real-time Frontend Update** — Conditions re-evaluate on every field change in the browser (always on)
- **Server-side Evaluation** — Conditions verified on submission; manipulation-proof
- **Debug Logging** — Plugin-level and per-field debug logging
- **Repeatable Group Integration** — Full per-row conditional logic inside Repeatable Group Field rows (requires Repeatable Group v1.6.2+ and ACL v1.3.54+); each row evaluates independently — including combined AND/OR conditions (v1.3.57+); new, duplicated and restored rows receive correct initial state automatically

---

Condition Types

1. Submission Count

Evaluates the number of completed submissions for the current form. Useful for capacity management, early bird pricing, and progressive messaging.

Available operators: `equals`, `not_equals`, `greater_than`, `less_than`, `greater_or_equal`, `less_or_equal`

Example use cases:

- First 100 registrants see early bird price
- Warning message after 80 submissions

- Waitlist option appears when form is full

---

2. Field Value

Evaluates the current value of another field in the form. Supports all field types with type-aware operator filtering.

Available operators by field type:

Field Type	Available Operators
Text / Textarea	all operators including `contains`, `is_empty`, `matches_regex`
Select (dropdown)	`equals`, `not_equals`, `is_empty`, `is_not_empty`
Radio group	`equals`, `not_equals`, `is_empty`, `is_not_empty`
Checkbox group	`contains`, `not_contains`, `is_empty`, `is_not_empty`

Checkbox note: A checkbox submission stores all checked values as a comma-separated string (e.g. `"apple,pear,plum"`). Use `contains` to test whether a specific option was checked. `equals` would only match if that single value is the only one selected.

Regex operators: `matches_regex` and `not_matches_regex` are available for text and textarea fields. The value field accepts a plain regular expression pattern without delimiters — the plugin adds them automatically.

Pattern	Example	Matches
`^[0-9]{4}$`	Postal code (4 digits)	`1234` ✓, `12345` ✗
`^\+?[0-9\s\-]{8,}$`	Phone number	`+36 1 234 5678` ✓
`^[^@\s]+@[^@\s]+\.[^@\s]+$`	Email format	`user@example.com` ✓
`^[A-Z]{2}[0-9]{6}$`	Passport number	`AB123456` ✓

Because frontend evaluation is always real-time, regex conditions re-evaluate on every keystroke — the user sees instant feedback as they type. Invalid regex patterns are handled gracefully: `matches_regex` returns false, `not_matches_regex` returns true.

Values `YES` and `NO` (fixed in v1.3.60): a condition value of exactly `YES` or `NO` now works like any other value. Earlier versions converted it internally, so such a condition was never met.

Value input: For select, radio, and checkbox fields, the condition value input automatically becomes a dropdown populated with the field's actual options. In multilingual forms the dropdown shows `base_value - Translated Label` — see Multilingual Support below.

---

3. Time-Based

Compares the current server time to a configured Unix timestamp. The admin interface includes a datetime picker that converts to Unix timestamp automatically.

Available operators: `equals`, `not_equals`, `greater_than`, `less_than`, `greater_or_equal`, `less_or_equal`

---

4. User-Based

Evaluates the current visitor's authentication status or identity.

User conditions:

- `is_logged_in` — visitor is authenticated (no value needed)
- `is_guest` — visitor is not authenticated (no value needed)
- `user_id` — compare against specific Joomla user ID
- `user_group` — compare against Joomla user group ID

Available operators (user_id / user_group): `equals`, `not_equals`, `greater_than`, `less_than`, `greater_or_equal`, `less_or_equal`

---

5. Combined Conditions

Groups multiple rules with AND or OR logic. Any condition type can be combined.

Operators: `AND` (all rules must be true) / `OR` (at least one rule must be true)

How combined rules are evaluated (v1.3.55+):

- `field_value` rules are re-evaluated live in the browser from the form's current values — the combined result changes as the user types or selects.
- `submission_count`, `time_based` and `user_based` rules cannot be evaluated in the browser; for these the result computed by the server at page load is used, rule by rule.
- The main Operator / Value settings of the CONDITION tab are ignored for combined conditions — only the rules and the AND/OR operator count.
- **Fixed in v1.3.60:** `user_based` rules inside a combined condition are evaluated with their own setting (`is_guest`, `is_logged_in`, user ID, user group). Earlier versions evaluated them with the field's main User Condition setting, so for example an `is_guest` rule was false for guests. If you built a workaround for this, review it after updating.

Warnings shown in admin:

- 0 rules configured — condition will always evaluate FALSE

- 1 rule — AND/OR has no effect; consider using a simple condition type

---

Action Types

Actions execute on other form fields when a condition evaluates TRUE or FALSE. Each conditional field has two independent lists: **Actions if TRUE** and **Actions if FALSE**.

Complete Action Reference

Action	Effect	Value
`show`	Makes the target field visible	—
`hide`	Makes the target field invisible	—
`enable`	Enables a disabled input	—
`disable`	Disables an input (greys out, prevents input)	—
`required`	Makes the field mandatory	—
`optional`	Removes mandatory requirement	—
`set_value`	Sets the field's value	plain text
`set_default_value`	Sets an editable default value — written only when the condition switches to this branch	plain text
`calculate`	Sets value using a math expression with field references	expression
`clear_value`	Empties the field's value	—
`set_html`	Replaces inner HTML of a display element	HTML allowed
`add_class`	Appends CSS class(es) to the field's **input element**	class name(s)
`remove_class`	Removes a CSS class from the field's **input element**	class name
`set_class`	Replaces all CSS classes on the field's **input element**	class name(s)
`add_block_class`	Appends CSS class(es) to the field's **block wrapper** (`.rsform-block-…`)	class name(s)
`remove_block_class`	Removes a CSS class from the field's **block wrapper**	class name

Action	Effect	Value
`set_block_class`	Replaces all CSS classes on the field's **block wrapper**	class name(s)
`add_style`	Adds inline CSS declarations to the field's **input element**	`prop: value; prop: value`
`remove_style`	Restores the original value of inline CSS properties on the **input element**	property names or the same declarations
`add_block_style`	Adds inline CSS declarations to the field's **block wrapper**	`prop: value; prop: value`
`remove_block_style`	Restores the original value of inline CSS properties on the **block wrapper**	property names or the same declarations
`add_option`	Adds a new option to a select/radio/checkbox field	`value|Label`
`remove_option`	Removes an option by value	dropdown
`show_option`	Makes a hidden option visible again	dropdown
`hide_option`	Hides an option from view	dropdown
`enable_option`	Re-enables a disabled option	dropdown
`disable_option`	Disables an option (greyed out, unselectable)	dropdown
`check_option`	Checks a specific checkbox option	dropdown
`uncheck_option`	Unchecks a specific checkbox option	dropdown
`select_option`	Selects a specific option in a select or radio field	dropdown
`call_function`	Calls a global JavaScript function when the condition changes state	`functionName` or `functionName|argument`

For actions marked **dropdown**: the value input automatically shows the target field's available options — no manual typing needed.

Action Availability by Field Type

The builder automatically shows only the relevant actions for the selected target field. The table below shows which actions are available for each field type:

Action	textbox / textarea	select	radio	checkbox	freetext / preview	button / submit
`show` / `hide`	Yes	Yes	Yes	Yes	Yes	Yes
`enable` / `disable`	Yes	Yes	Yes	Yes	—	Yes
`required` / `optional`	Yes	Yes	Yes	Yes	—	—
`set_value` / `clear_value`	Yes	Yes	Yes	Yes	—	—
`set_default_value`	Yes	—	—	—	—	—
`calculate`	Yes	—	—	—	—	—
`set_html`	—	—	—	—	Yes	—
`add_class` / `remove_class` / `set_class`	Yes	Yes	Yes	Yes	Yes	Yes
`add_block_class` / `remove_block_class` / `set_block_class`	Yes	Yes	Yes	Yes	Yes	Yes
`add_style` / `remove_style`	Yes	Yes	Yes	Yes	Yes	Yes
`add_block_style` / `remove_block_style`	Yes	Yes	Yes	Yes	Yes	Yes
`add_option` / `remove_option`	—	Yes	Yes	Yes	—	—
`show_option` / `hide_option`	—	Yes	Yes	Yes	—	—
`enable_option` / `disable_option`	—	Yes	Yes	Yes	—	—
`check_option` / `uncheck_option`	—	—	—	Yes	—	—
`select_option`	—	Yes	Yes	—	—	—
`call_function`	Yes	Yes	Yes	Yes	Yes	Yes

For `call_function` the target field is optional — leave it empty if the function does not need one.

For unknown or third-party field types all actions are shown as a safe fallback.

Notes on Specific Actions

`set_html` vs `set_value`
`set_value` inserts plain text into an input. `set_html` replaces the inner HTML of a display element (`freetext`, `preview`) — HTML tags, bold, line breaks, icons, and links are rendered. Using `set_html` on an input field has no visible effect.

`set_value` vs `set_default_value` (v1.3.61)
`set_value` is re-applied on every evaluation. While the condition holds, the field keeps the configured value: if the visitor types something else, it is set back at once. Use it for values that must not change (a fixed price, a code).
`set_default_value` writes the value only when the condition **switches** to the branch that contains it. After that the visitor can edit the field freely; typing in it or changing other fields does not reset it. The next time the condition switches to that branch, the value is written again (the visitor's edit is replaced).

- **On page load** (and for a new or restored Repeatable Group row) the value is written only if the field is **empty**. A default value set in RSForm! Pro, and the value sent back after a failed submission, are kept.
- **Duplicated Repeatable Group rows** keep the value copied from the original row.
- Available for text-type fields: textbox, textarea, calendar, phone number. For select, radio and checkbox fields use RSForm! Pro's own default selection.
- Example: *Actions if TRUE* → `quantity` → Set default value (editable) → `10`. Leave *Actions if FALSE* empty if the visitor's value should stay when the condition becomes false, or add `clear_value` to empty it.

`set_class` vs `add_class` (input element)
`set_class` replaces all existing classes on the input element — use when switching between alert colour styles (e.g. `alert-success` to `alert-danger`). `add_class` appends — use when adding a modifier without removing base classes.

`set_block_class` / `add_block_class` / `remove_block_class` (block wrapper)
The `_block_class` variants target the `.rsform-block-{fieldname}` wrapper div instead of the input element itself. Use these to apply layout classes, highlight an entire field row, or add Bootstrap utility classes to the wrapping container.

`add_option` value format
Accepts `value|Display Label` (pipe-separated) or just `value` when label equals value. Duplicates are ignored automatically.

`check_option` / `uncheck_option`
Checkbox-only. Each action affects one specific option. Use multiple actions to check or uncheck several options independently in a single rule.

`select_option`
For select and radio fields. Sets one specific option as selected; all others are automatically deselected.

`calculate`
References other field values using `{fieldname}` placeholders. Uses a custom safe math parser — `eval()` is never called. Example: `{quantity} * {unit_price} * 1.27`

`add_style` / **`remove_style`** / **`add_block_style`** / **`remove_block_style`**
The value is a list of CSS declarations separated by semicolons, e.g. `background-color: #fde2e2; border: 1px solid #c00`. A semicolon inside parentheses or quotes does not split the list, so values such as `url("data:...;base64,...")` stay intact.

- On the first `add_style`, the plugin remembers the property's original inline value. `remove_style` restores that original value instead of simply deleting the property — inline styles set by your template or other plugins survive.
- For removal, property names are enough (`background-color; border`), but you can also paste the exact same declarations used for adding. This lets you copy one value into both the TRUE and FALSE lists.
- `remove_style` only undoes changes made by the plugin; it does not remove inline styles it did not set.
- **Ownership (v1.3.57+):** when several conditional fields style the same property of the same element (e.g. range-based colouring: green / amber / red), only the conditional field that set the value last can restore it. The FALSE branch of one rule therefore does not wipe the colour just applied by another rule.

Custom JavaScript — `call_function`

The `call_function` action connects the plugin's condition engine to your own JavaScript. It is intended for things the built-in actions cannot do: updating a map or chart, sending an analytics event, loading data, toggling a third-party widget.

Configuration

Setting	Value
Target field	Optional. If selected, the function receives it as `ctx.targetField`.
Action	Call JavaScript function
Value	`functionName` — or `functionName|argument` to pass a fixed text argument

The function name can be a plain global name (`onVipChange`) or a dotted namespace (`MyApp.onVipChange`). In the latter case `this` inside the function is `MyApp`.

Put the action into **Actions if TRUE** to run it when the condition becomes true, into **Actions if FALSE** to run it when it becomes false — or into both lists and branch on `ctx.conditionMet`.

When the function is called

- **Only on state change.** Conditions are re-evaluated on every keystroke and every change. The function runs only when the result switches from false to true or from true to false — not on every re-evaluation.
- **Once on first evaluation.** On page load, and when a Repeatable Group row is added or duplicated, the function runs once with `ctx.isInitial === true`, so it can set up the initial state. If you do not need this, start the function with `if (ctx.isInitial) return;`.
- **Per row in Repeatable Groups.** State is tracked separately for every row; changing row 2 calls the function for row 2 only.
- **In list order.** The call happens in its place among the other actions, so a function listed after a `set_value` already sees the new value.

The context object

The function receives a single argument:

Property	Type	Description
`conditionMet`	boolean	New state of the condition
`previousState`	boolean | null	Previous state; `null` on first evaluation
`isInitial`	boolean	`true` on page load / new Repeatable Group row
`conditionalId`	string	ID of the Advanced Conditional Logic field
`conditionalEl`	HTMLElement	The conditional field's hidden input
`form`	HTMLFormElement	The form
`scopeEl`	HTMLElement | null	The Repeatable Group row (`.rg-row`), or `null` outside groups
`rowIndex`	number | null	0-based index of the row within its group
`triggerFieldName`	string	Name of the (first) trigger field
`triggerFieldNames`	string[]	All trigger fields (several for combined conditions)
`triggerField`	HTMLElement | null	The trigger field element — in the same row, inside groups
`targetFieldName`	string	Target field selected in the builder (`''` if none)

Property	Type	Description
`targetField`	HTMLElement | null	The target input element
`targetContainer`	HTMLElement | null	The target's `.rsform-block` wrapper
`arg`	string	Text after the `|` in the action value (`''` if none)

Example

Add this to the form's JavaScript (RSForm! Pro → Form → Properties → CSS and JavaScript), or to your template:

```
<script>
window.onVipChange = function (ctx) {
   if (ctx.isInitial) return;             // skip page load
   console.log('VIP is now', ctx.conditionMet ? 'on' : 'off',
               'row', ctx.rowIndex, 'arg', ctx.arg);
};
</script>
```

Action: *Actions if TRUE* and *Actions if FALSE* → Call JavaScript function → `onVipChange|ticket`.

Safety and error handling

- The name is looked up on `window` — `eval()` and `new Function()` are never used. Only identifier characters and dots are accepted; the segments `__proto__`, `prototype` and `constructor` are rejected. A trailing `()` typed by mistake is ignored.
- If the name is invalid or the function does not exist, the action is skipped with one console warning (not one per state change).
- If the function throws an error, it is logged to the console and the remaining actions still run.
- The function must exist when the form initialises (normally on `DOMContentLoaded`). Functions defined inline or in normal/deferred scripts are fine; a function defined in an `async` script may not be available yet.
- Calling `applyAction(field, 'call_function', 'name', id, scope)` directly from your own code calls the function unconditionally (with `conditionMet: null`).

Admin Condition Tester

Each conditional field displays a **Test** button in the admin editor. Clicking it immediately evaluates the configured condition against live data — without needing to open the frontend form.

Supported condition types:

Condition type	Testable in admin?	Notes
`submission_count`	✅ Yes	Queries the actual submission count
`time_based`	✅ Yes	Compares against current server time
`user_based`	✅ Yes	Uses the currently logged-in admin user
`field_value`	⚠️ No	Requires a live form with POST data
`combined`	⚠️ No	Requires a live form with POST data

The result is shown inline next to the button: ✅ **TRUE** or ❌ **FALSE**, with the current and target values displayed for comparison. This allows quick verification of time windows and submission thresholds without submitting test entries.

Repeatable Group Field Integration

The Advanced Conditional Logic plugin is fully compatible with the **Repeatable Group Field** plugin (plg_system_repeatablegroup) as of **ACL v1.3.54** and **Repeatable Group v1.6.2** (combined AND/OR conditions per row: ACL v1.3.57+). Each row in a repeatable group evaluates its own conditional rules independently.

How It Works

In a standard RSForm! Pro form, field names follow the pattern `form[fieldName]`. The Repeatable Group plugin rewrites these to `form[groupId][N][fieldName]` during row indexing. Prior to v1.3.53, the ACL engine searched for the standard pattern only — meaning the trigger field was never found within a Repeatable Group row, and show/hide conditions never fired after the initial page load.

From v1.3.53, the ACL engine detects when a trigger field exists inside a `.rg-row` container and automatically switches to **row-scoped evaluation**: each row is evaluated against its own trigger field value, and actions are applied only to target fields within the same row. From v1.3.57 this also applies to **combined** conditions: if any `field_value` rule of a combined condition refers to a field inside a repeatable group, the whole combined condition is evaluated row by row.

Example: In a Participant registration form with multiple rows, checking "Special dietary requirement?" in row 2 shows the "Dietary details" field in row 2 only — row 1 and row 3 are unaffected.

Setup

No special configuration is required. Configure the condition and actions exactly as you would for a non-repeatable form field. The ACL plugin detects the Repeatable Group context automatically at runtime.

Setting	Value
Condition type	Field value
Trigger field	The checkbox / select / radio field inside the repeatable group
Condition	e.g. `contains` → `yes`
Actions if TRUE	`show` → the dependent field (must be in the same row)
Actions if FALSE	`hide` → the dependent field

Row lifecycle

The Repeatable Group plugin notifies the ACL engine whenever a row is created, duplicated, or restored from session backup. The ACL immediately evaluates the initial state of that row — so a newly added row with an unchecked checkbox will correctly hide its dependent field without any user interaction.

This integration requires the ACL plugin to expose its `window.ConditionalFieldGroup.evaluateScope(rowEl)` API (available from v1.3.53). The Repeatable Group plugin calls this method with the new row element after setup is complete.

Rows restored after a failed submission (v1.3.60): when a submission fails validation, the Repeatable Group plugin restores every row with its values. From v1.3.60 the restored checkbox and radio selections are always kept, whichever of the two plugins' scripts the page loads first, and the per-row rules follow the restored values. (With earlier versions, depending on script order, restored checkbox/radio trigger fields could be reset to their defaults.)

`call_function` actions follow the row lifecycle too: a new or duplicated row is treated as a first evaluation (`ctx.isInitial === true` , with `ctx.scopeEl` and `ctx.rowIndex` set to that row), and a deleted row's state is discarded with it.

Pages without a repeatable group

From Repeatable Group v1.7.15, that plugin loads its scripts only on pages that contain a repeatable group. Advanced Conditional Logic does not depend on them: on a page with conditional rules but without a repeatable group, every rule — including `call_function` — works as usual. The ACL only recognises repeatable rows from the page markup; it never requires the Repeatable Group script.

Limitations

- **Same-row only:** The trigger field and the target field must be in the **same repeatable group row**. A trigger in row 1 cannot affect a field in row 2, or a field outside the group.
- **Field value and combined conditions only:** Row-scoped evaluation applies to `field_value` conditions and — from v1.3.57 — to `combined` conditions whose `field_value` rules refer to fields inside the group. Other types (`submission_count` , `time_based` , `user_based`) evaluate globally.

- **Server-side scope:** PHP-side evaluation does not currently support per-row scope. The initial `hide` CSS class is applied globally to the field's `.rsform-block` wrapper — which works correctly for the initial page state — but server-side condition results do not vary per row.
- **Requires both plugins updated:** Repeatable Group v1.6.2+ and ACL v1.3.54+ are both required for this integration. With older versions of either plugin, ACL conditions inside repeatable groups affect only the first row.

Evaluation Timing — When Conditions Are Checked

The plugin evaluates conditions in two independent layers. Understanding this helps you predict exactly when a condition result becomes visible to the user.

Frontend Layer (JavaScript) — real-time, in the browser

The `conditional.js` engine listens to `change` and `input` events on all form fields. When a field changes, it re-evaluates all conditions of the form in execution order and immediately executes the resulting actions in the DOM — no page reload, no server round-trip. Actions are re-applied on every evaluation, except `call_function` and `set_default_value`, which run only when a condition changes state.

When exactly does each field type trigger re-evaluation:

Field type	Trigger	Timing
Textbox / Textarea	`input` event	On every keystroke, as the user types
Select (dropdown)	`change` event	Immediately on selection
Radio group	`change` event	Immediately on click
Checkbox group	`change` event	Immediately on each check/uncheck
Calendar / Date	`change` event	Immediately on date selection

The frontend engine always re-evaluates on every field change. Conditions are checked in real-time as the user interacts with the form — no configuration needed.

Backend Layer (PHP) — on every form submission

The PHP `checkConditionRule()` method runs server-side every time the form is submitted. It re-evaluates all conditions independently of the JavaScript result and executes actions (e.g. blocking submission, modifying values) based on the server-side outcome.

This layer is **manipulation-proof** — a visitor cannot bypass it by disabling JavaScript or modifying the page DOM. It is also the layer that handles conditions for users who have JavaScript disabled entirely.

Summary

	Textbox / Textarea	Select / Radio / Checkbox
JS	every keystroke	on every selection change
PHP	on form submit	on form submit

The JavaScript layer provides the **user experience** — instant visual feedback. The PHP layer provides **correctness and security** — the final authoritative result that cannot be bypassed.

Initial State — Flash-free Field Hiding

When a page loads, fields that should start hidden might briefly appear before JavaScript runs — causing a flash of unwanted content. The plugin eliminates this automatically.

How it works: The plugin evaluates conditions server-side during page rendering. For each `hide` action in the active action list, it writes a small inline `<style>` rule for that field's `.rsform-block-{fieldname}` wrapper, directly before the form field:

```
.rsform-block-fieldname:not(.conditional-shown) { display: none !important; }
```

This rule is in the HTML before the browser paints the form, so the field is never visible on first render. When JavaScript loads and re-evaluates conditions, `show` adds the `.conditional-shown` class (which releases the rule) and `hide` adds `.conditional-hidden`.

The plugin never forces `display: block` — when a field is shown again, it gets back its natural display value, so flex- and grid-based form layouts are not broken (v1.3.54+).

No configuration needed. Set up the IF TRUE / IF FALSE action lists correctly — `hide` in the appropriate branch — and the initial state is handled automatically.

Multilingual Support

The plugin is fully compatible with RSForm! Pro's multilingual form system.

How RSForm! Pro Stores Multilingual Options

RSForm! Pro stores field options (ITEMS) in two places:

- `#__rsform_properties` — the base values, language-independent. These are what get stored in form submissions.
- `#__rsform_translations` — translated labels for each language, keyed by `{ComponentId}.ITEMS`.

The plugin always uses the **base value** for condition evaluation and storage. The translated label is only used for display.

Option Dropdowns in Admin

When configuring a field value condition or an option-level action for a multilingual form, the dropdown shows:

```
base_value - Translated Label
első - regular
második - VIP
```

This means you configure conditions using the base value (`első` , `második`) — which is what RSForm! Pro stores regardless of language — while the translated label helps you identify which option is which. The condition works correctly for visitors in any language.

Caption Translation

Field names in the action builder display translated captions when available. If the base CAPTION property is empty (common in multilingual forms where only the translation is filled in), the plugin retrieves the translated caption automatically.

Interface Language

Complete Hungarian and English translations are included for all admin UI labels, operator names, action names, warnings, and tooltips.

Field Configuration

GENERAL Tab

Property	Description
Name	Internal field identifier (no spaces, ASCII only)
Caption	Display label (can be left empty)
Description	Help text shown below the field

CONDITION Tab

Property	Description
Condition Type	`submission_count` , `field_value` , `time_based` , `combined` , `user_based`

Property	Description
Operator	Comparison operator (depends on condition type)
Value	The value to compare against
Field Name	Source field for `field_value` condition
Combined	Visual builder for AND/OR combined rules
User Condition	Sub-type for `user_based` conditions
User Value	Value for `user_id` / `user_group` comparison

ACTIONS Tab

Property	Description
Actions if TRUE	Ordered list of actions executed when condition is met
Actions if FALSE	Ordered list of actions executed when condition is not met

ADVANCED Tab

Property	Description
Enable Logging	Log condition evaluations to Joomla debug log (category `rsform.conditional`; works from v1.3.60)
Execution Order	Priority when multiple conditional fields exist (lower = earlier; default 100)

There is no separate real-time switch or cache setting: frontend re-evaluation is always on, and conditions are evaluated fresh on every page render.

Plugin Global Settings

Found under **Extensions → Plugins → System - Advanced Conditional Logic for RSForm! Pro**

Setting	Description
Load frontend stylesheet	Load the plugin's frontend CSS (`conditional-frontend.css`) with transitions, animations, and `.conditional-hidden` / `.conditional-shown` styles. Disable if your template already handles these styles.

Where the plugin's CSS and JavaScript are loaded

The frontend files — `conditional.js` and, if enabled, `conditional-frontend.css` — are added **only to pages that display a form containing at least one published Advanced Conditional Logic field**. This works the same way whether the form is shown as a component page, in an RSForm! Pro module, embedded in an article, or in a `tmpl=component` / modal view. All other pages (home page, articles, shop pages, forms without conditional rules) get nothing from this plugin.

What this means for you:

- **Your own CSS or JavaScript on other pages.** `window.ConditionalFieldGroup` and the plugin's CSS classes exist only on pages with a conditional form. If your template or a custom script uses them elsewhere, check first: `if (window.ConditionalFieldGroup) { … }`.
- **Page optimisers (combine, defer, async, move to footer).** Supported. If an optimiser loads the script late, after the page has finished loading, the plugin initialises immediately when the script runs. If the script ends up on the page twice (for example, once in a combined bundle and once on its own), the second copy does nothing.
- **Forms opened in a popup.** A popup that shows the form page in an iframe (for example a `tmpl=component` link) works as usual. A popup script that inserts the form's HTML into the already loaded page is not supported: the conditional rules start once per page load, so a form added afterwards is not picked up. Use an iframe-based popup for conditional forms.
- **Safety net.** If a form is rendered too late for the page head (for example by another plugin after the page has been assembled), the plugin adds any missing file just before `</body>`. Such tags carry a `data-acl-fallback="1"` attribute. You don't need to do anything about it; it can help when reporting a problem.

Technical Specifications

Requirements:

- Joomla 4.x, 5.x, 6.x
- RSForm! Pro 3.x+
- PHP 7.4+
- MySQL 5.7+
- Modern browser with JavaScript enabled
- For Repeatable Group integration: Repeatable Group Field plugin v1.6.2+ (recommended together: Repeatable Group v1.7.15+ with ACL v1.3.60+)

Installation:

1. Download plugin ZIP file
2. Joomla Admin → Extensions → Install

3. Upload ZIP file
4. Enable: System - Advanced Conditional Logic for RSForm! Pro plugin
5. Clear Joomla cache

Usage:

1. RSForm! Admin → Components → Forms → Edit Form
2. Add new field: Advanced Conditional Logic
3. Configure CONDITION and ACTIONS tabs
4. Save form
5. Test on frontend

Package Contents

```
plg_system_conditional.zip
├── conditional.php               # Main plugin file
├── conditional.xml               # Installation manifest
├── script.php                    # Install/update script
├── admin/helpers/fields/
│   └── conditionalgroup.php      # RSForm! field class
├── media/css/
│   ├── conditional.css           # Admin styles
│   └── conditional-frontend.css  # Frontend styles (transitions, animations,
conditional-hidden)
├── media/js/
│   ├── conditional.js            # Frontend evaluation engine (incl. call_function)
│   ├── conditional-builder.js    # Action builder
│   ├── conditional-combined.js   # Combined condition builder
│   ├── conditional-fieldselector.js  # Field value condition widget
│   ├── conditional-timepicker.js # Time-based datetime picker
│   ├── conditional-userpicker.js # User-based condition widget
│   └── conditional-test.js       # Admin condition tester
├── sql/mysql/
│   ├── install.sql
│   ├── uninstall.sql
│   ├── update_1_1_0.sql
│   └── update_1_3_5.sql
├── language/en-GB/
└── language/hu-HU/
```

License

License: GNU General Public License v2.0 or later
Commercial Use: Permitted under GPL
Modification: Allowed under GPL
Distribution: Allowed under GPL

Contact & Support

Developer: László Blahó
Website: https://blaho.hu
Extensions: https://extensions.blaho.hu
Email: extension@blaho.hu