Create and update report templates

Last Modified on 09/25/2026 2:00 pm EDT

A report template combines a Word document with data sources configured in DevResults. The document controls the layout. Placeholders insert values, repeat sections, and include or omit content when someone generates a report.

This guide walks through creating a template, updating an existing one, and writing placeholders. It uses the two downloadable sample templates introduced below. If you only need to run a report, see **Generate a report from a template**.

Before you start

Your site must have Report Templates enabled. You need Report Templates edit permission to upload and maintain templates. Users who generate reports need view permission. If you can generate reports but cannot see **New template** or **Edit**, ask your administrator to check your access.

Use a `.docx` file for the walkthrough. Word templates can generate Word files or PDFs. The upload control also accepts `.pptx` and `.xlsx`, but acceptance alone does not establish that a report will render correctly. Check those formats with DevResults support before building a new template around them. The examples and screenshots here were checked with Word templates.

This guide uses **activity**. Your site may call the same record a **project** or **award**. The screenshots show **Project**; the placeholder key remains whatever you configure, such as `activity`.

Create your first template

1. Prepare the document

Start with the **sample report note**, or create a Word document with this content:

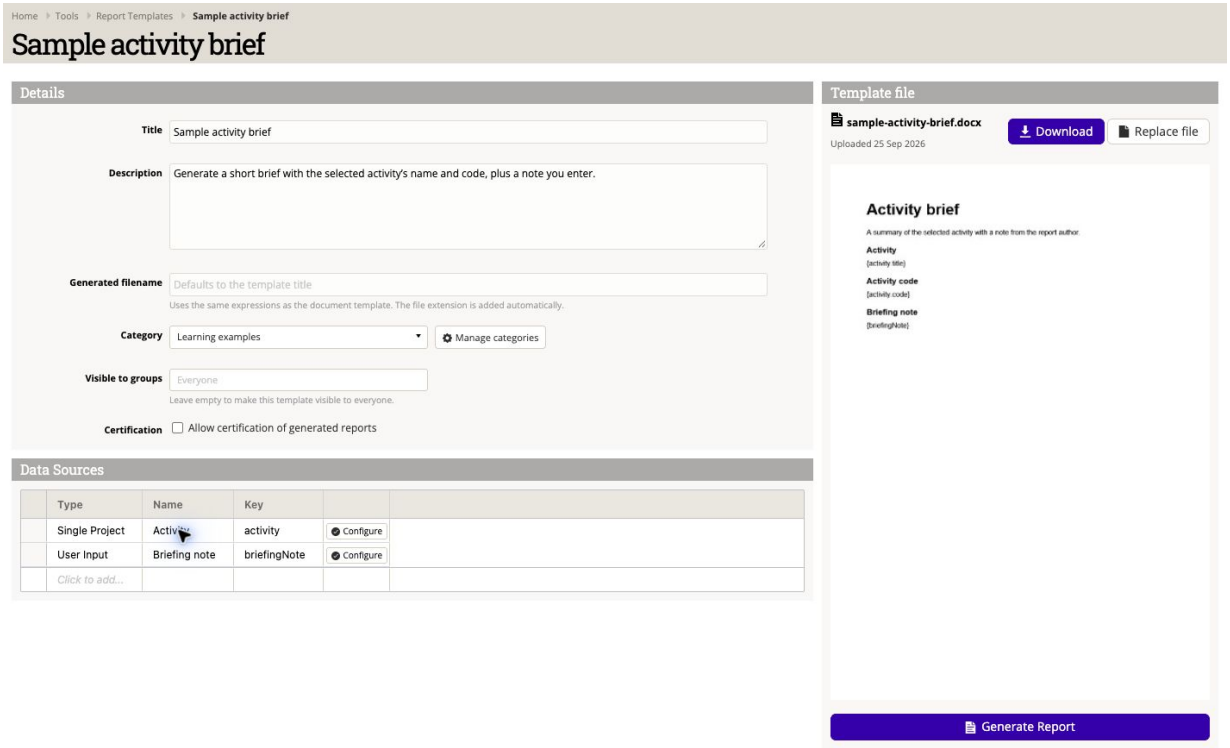
```
{subject}

Summary
{summary}

Next step
{nextStep}
```

Apply headings, fonts, margins, tables, headers, and footers in Word. The inserted text takes the formatting of the placeholder. Type each complete placeholder as ordinary text, including the braces. Save the file as `.docx`.

Plan where each value comes from before adding more placeholders. Text that the user types needs a **User Input** source. An activity name needs an activity source. A list of query results needs a query source and a loop.

2. Upload the file

1. Open **Tools > Report Templates**.
2. Click **New template** and choose the `.docx` file.
3. On the detail page, replace the initial filename-based title with a useful report title.
4. Add a description that explains the report's purpose and any inputs the user needs.
5. Choose a category, or type a new category name.



The activity brief has two data sources. Its Word file uses the keys shown in the grid.

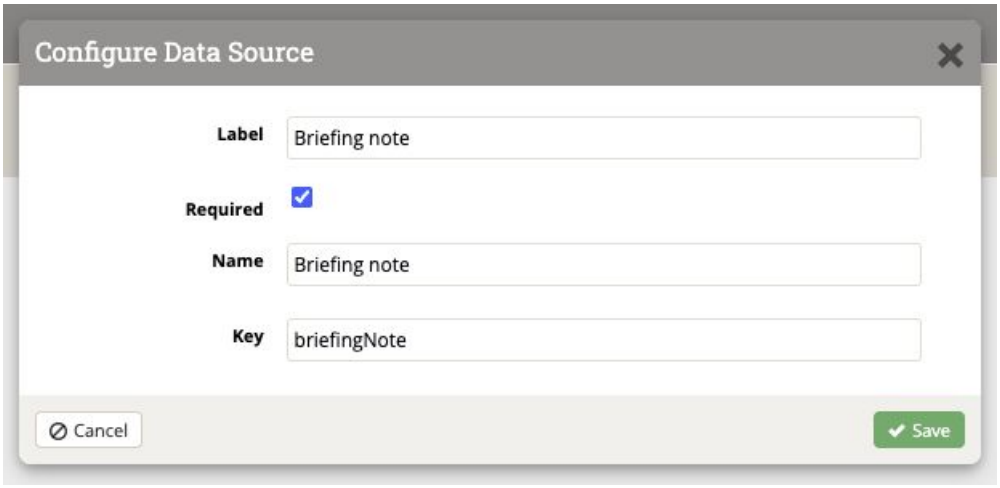
3. Add data sources

In **Data Sources**, use **Click to add...** to choose a source type. Give the source a descriptive **Name** and a unique **Key**. Click **Configure**, complete the source-specific settings, and save the configuration.

For the report note, add these three **User Input** sources:

Name	Key	Label	Required
Subject	`subject`	Subject	Yes
Summary	`summary`	Summary	Yes
Next step	`nextStep`	Next step	Yes

The **Name** identifies the source to editors. The **Label** is the prompt users see during generation. The **Key** connects the source to placeholders. `{nextStep}` works with the key `nextStep`; `{NextStep}` does not.



For the activity brief, the required Briefing note prompt supplies `{briefingNote}`.

Keep keys stable after people start using a template. Changing a label or source name does not require changing the document. Changing a key requires updating every placeholder that uses it, including expressions in the generated filename.

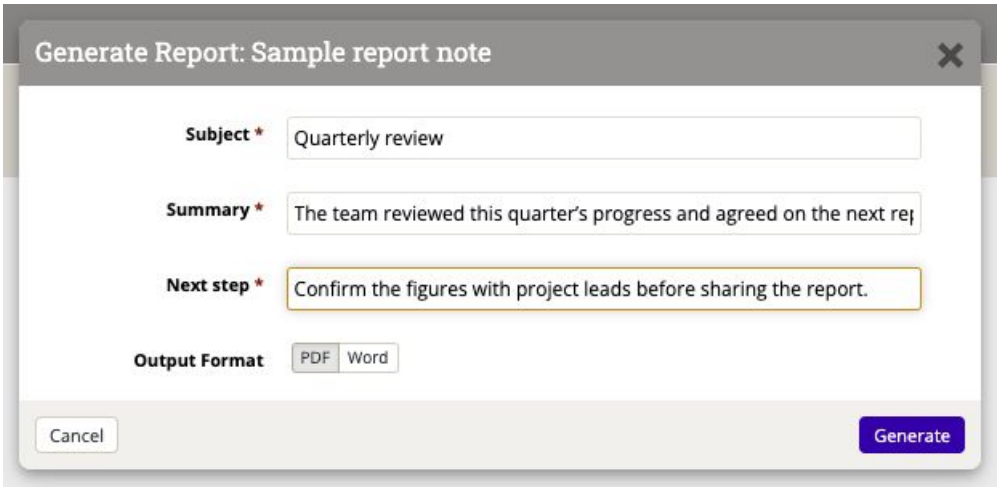
4. Set visibility and output details

Setting	How to use it
Title	Use a title that tells users which report they will get. It also supplies the default output filename.
Description	Explain the purpose and any preparation needed before generation.
Generated filename	Leave blank to use the title, or enter text and value placeholders. DevResults adds the extension.
Category	Group related templates in the gallery. Categories organize templates; they do not restrict access.
Visible to groups	Leave empty for everyone who has Report Templates access, or choose the groups that should see the template.
Certification	Enable only when your reporting process calls for it. Eligible users can certify, and placeholders can display the certification details.

Detail fields save automatically. Configuration dialogs have their own **Save** button. There is no separate draft/publish step for a template, so test changes on a test site or a separate test template before replacing a shared source file. A category named “Draft” does not make a template private.

5. Generate and inspect a report

Click **Generate Report**, complete the inputs, and generate a PDF. Generate Word as well when users need editable output. Open each result and check the text, filename, page breaks, tables, headers, and footers.



Test required fields by leaving one empty; **Generate** should remain unavailable. For more complex templates, also test empty lists, missing optional values, long text, several repeated rows, and every condition. If the template uses certification, test both certified and uncertified output with an eligible user.

A successful download does not prove that every placeholder is correct. Search the result for unexpected braces and check that values match the selected records. A missing expression can remain visible as its original placeholder.

Add activity data

The **sample activity brief** introduces a selected record. It contains:

```
Activity brief
{activity.title}
Activity code: {activity.code}

Briefing note
{briefingNote}
```

Add a **Single Activity** source (called **Single Project** in the screenshots), with Name `Activity` and Key `activity`. Configure the activity filters as needed. Add a required **User Input** source with Key `briefingNote` and Label `Briefing note`.

During generation, the user selects an activity and enters a note. If a report is opened from an activity page, that activity may already be supplied. Always check the generation context before comparing the output.

Choose data sources

Each source contributes a value under its key. A template can have at most one primary source. It can have secondary sources without a primary source, as the report note does.

Primary source	Input during generation	Value
All Activities	None	An array of activities available to the user.
All Indicators	None	An array of indicators available to the user.
Single Activity	Activity	One activity, including assigned indicators and report metadata.
Single Activity Reporting Period	Activity and reporting period	A reporting-period record with activity, period, indicators, narratives, and calculated progress.
Single Indicator	Indicator	One indicator object.

Single Activity sources can configure the available activity filters and initial statuses. These settings affect the choices

shown during generation; they do not create new activity data.

Secondary source	Configuration	Value
Custom Query	Select the query.	An array of rows with the query's column names.
Data Table	Select the table.	An array of rows with friendly column labels.
User Input	Set Label and Required.	The entered text, or `null` when empty. Checkbox values may occur in templates configured through other tooling; the current configuration dialog does not expose an input-type setting.
Contact	Set label, group, required status, activity scoping, and whether Other is allowed.	A contact with `displayName` and optional `title` and `email`, or `null`. Other supplies only `displayName`.

A selected activity from the primary source can also provide context for secondary sources, such as an activity-scoped contact picker. Column names and record fields depend on the source. The syntax reference below gives common paths; it is not a complete dictionary of every site's data.

Update an existing template

1. Open **Tools > Report Templates**, open the template's actions menu, and choose **Edit**.
2. Under **Template file**, click **Download** to start with the currently installed source. Keep a copy of this file before editing.
3. Edit the downloaded file in Word. For wording or layout changes, keep existing keys and placeholders intact. For new data, plan the corresponding data-source changes.
4. Use **Replace file** to upload the revision. Replacement keeps the template record and its settings, including configured sources and visibility.
5. Update data-source configurations and the generated filename if the new document requires them. Check each changed key against the document.
6. Generate representative reports in Word and PDF, and compare the changed sections with the previous version.

Replacing the source affects future reports. Reports already downloaded or saved to Documents retain their existing content. To recover from an incorrect replacement, upload the saved previous source and restore any settings you changed. Keep your own source versions; do not rely on the template page as a version history.

Maintain categories and remove templates

Use **Manage categories** to rename or reorder categories. Deleting a category leaves its templates uncategorized. Use visibility groups to control who can see a template.

To remove a template, use **Delete** in its gallery actions or **Delete this template** on its detail page, and complete the confirmation. Download the source first if you may need it again. Removing the template does not remove reports users already downloaded or saved to Documents.

Placeholder syntax reference

The following reference covers value placeholders, expressions, filters, loops, conditions, layout controls, special values, certification, shared helpers, and filename expressions. The final appendix describes helpers specific to the INL catalog so authors can recognize them in existing files without treating them as general reporting functions.

Rendering model

Every configured data source contributes one top-level value under its Key. If a Single Activity source has key `award` and a Custom Query source has key `budgetRows`, the render scope begins like this:

```
{
  "award": { "title": "Example activity", "code": "EX-01" },
  "budgetRows": [
    { "Year": 2025, "Budget": 125000 },
    { "Year": 2026, "Budget": 150000 }
  ]
}
```

DevResults then adds shared helpers, special values, certification data, and INL helpers. A template can combine all of them:

```
{award.code}: {award.title}
Generated {_today_}
{#budgetRows}{Year}: {Budget | number:0}
{/budgetRows}
```

Tags are evaluated when the user selects **Generate**. The Office file supplies the layout and formatting; tags supply values and control which Office structures repeat or remain.

Value tags

Wrap an expression in braces to insert its value.

```
{title}
{award.title}
{award.primaryOrganization_Title}
{rows[0].Amount}
{row['Column with spaces']}
```

Property names and data-source keys are case-sensitive. Custom-query and data-table fields often preserve human-readable column names, including spaces and punctuation, so bracket notation is sometimes required.

Values are XML-escaped before insertion. A string containing `&` or `<` appears as text rather than becoming Office XML.

Dot notation and the current item

Inside a loop, unqualified names refer to the current item. `this` is the current item itself. Use `{this.name}` for an object property. Arrays of primitive strings or numbers are not supported by the current scope wrapper; use arrays of objects instead. For the disaggregation helper, print `valuesText` rather than looping over its string-valued `values` array. The default Docxtemplater shorthand `{.}` is not accepted by the AngularJS parser.

```
{#indicators}
{code}: {title}
{indicatorHasHistory(this) ? 'Has history' : 'No history'}
{/indicators}
```

If a name is not present on the current item, DevResults checks parent loop scopes and finally the root scope. This makes root values available inside nested loops:

```
{reportTitle}
{#groups}
{#members}{name} – {reportTitle}{/members}
{/groups}
```

Prefer explicit paths when the same property name exists at more than one level.

Expressions

Tags use AngularJS expression syntax. The commonly useful forms are:

Form	Example	Result
Property access	`{award.title}`	Reads a nested value.
Bracket access	`{row['Total Cost']}`	Reads a property whose name is not a simple identifier.
Comparison	`{amount >= 100}`	Produces a Boolean value.
Arithmetic	`{accepted + returned}`	Produces a calculated number.
Ternary	`{name ? name : 'No response provided.'}`	Chooses one of two values.
String method	`{referenceNumber.replace('/', ' ')}`	Calls a safe method available on the value.
Array length	`{rows.length}`	Returns the number of items.
Array literal	`{condition ? [this] : []}`	Produces one item or no items for a structural condition.
Assignment	`{#(count = rows.length)}...{/}`	Stores a value on the current scope for later tags. Use sparingly.

Use quoted string literals for fixed text inside expressions. Existing templates use single quotes because they are easier to read inside Office content.

```
{status == 'Approved' ? 'Ready' : 'In progress'}
```

Use parentheses to group expressions. Comparison operators (`<` , `<=` , `>` , `>=`), equality operators (`==` , `!=` , `===` , `!==`), Boolean operators (`&&` , `||` , `!`), and arithmetic (`+` , `-` , `*` , `/` , `%`) are supported. Type these as ordinary text in Word; Word handles XML escaping. If editing raw Office XML, escape XML characters correctly.

```
{(accepted + returned) | number:0}
{score < 20 && approved ? 'Review' : 'Ready'}
```

This is AngularJS expression syntax, not general JavaScript. Do not use arrow functions, `new` , variable declarations, optional chaining, or nullish coalescing. Use `value == null ? fallback : value` for a null-aware fallback. Object literals contain braces that collide with the default tag delimiters; use a configured value or helper instead.

Filters

Use a pipe to pass a value through an AngularJS filter.

```
{amount | number:2}
{_now_ | date:'yyyy-MM-dd HH:mm:ss'}
```

The `number` filter adds locale-aware grouping and decimal places. The `date` filter accepts AngularJS date-format tokens. Registered DevResults filters can also be used, but a template should prefer the shared helpers documented below when one exists because those helpers are part of the report-rendering contract.

Filters can use values from a parent scope:

```
{#items}{amount | number:decimalPlaces}{/items}
```

AngularJS `$parse` cannot reliably index the result of a filtered array in this renderer. Do not write `(rows | filter:criteria)[0]` . Use `findBy(rows, 'property', value)` for exact lookups or add a provider/helper when the required shaping is more complex.

The standard AngularJS filters available to expressions include `number`, `currency`, `date`, `uppercase`, `lowercase`, `json`, `limitTo`, `filter`, and `orderBy`. Arguments follow colons, and filters can be chained. Examples: `{title | uppercase}`, `{amount | number:2}`, and `{_now_ | date:'yyyy-MM-dd':'UTC'}`. Date-filter tokens differ from the `formatDate` helper's tokens. Additional site filters are not a stable general-template contract.

Loops and conditions

The same block syntax handles arrays and truthy conditions.

Repeat an array

```
{#items}
{name}: {amount | number:0}
{/items}
```

The content between the tags repeats once for each item. The closing tag can repeat the expression name, as above, or use the shorthand `{/}`.

```
{#items}{name}{/}
```

Use named closing tags for long or nested blocks. They make mismatches easier to find.

Show content conditionally

A block around a non-array expression renders once when the expression is truthy and not at all when it is falsy.

```
{#email}
Email: {email}
{/email}

{#amount > 99}
Large award
{/amount > 99}
```

A block is omitted for `false`, `null`, `undefined`, an empty string, zero, or an empty array. An empty array is truthy in ordinary JavaScript expressions but has no iterations in a template loop.

Inverse blocks

Prefix the expression with `^` to render only when the value is falsy or the array is empty.

```
{#items}
...rows...
{/items}
{^items}
No items were found.
{/items}
```

Inverse blocks are useful for an empty-state row or a fallback paragraph.

One-or-zero array conditions

When a structural loop must repeat a row, cell, or run exactly once or not at all, return a one-item array or an empty array from a ternary:

```
{#finalReport ? [awardReportingPeriod] : []}
Final-report-only content
{/}
```

This pattern is especially useful in tables because the loop has an explicit current item and a predictable repeat count.

Nested loops

Loops can be nested. Close each named block at the correct level.

```
{#groups}
{title}
{#members}
{name}
{/members}
{/groups}
```

Office layout rules

Tags are plain text, but they control XML structures inside the Office package. The placement of control tags therefore affects the result.

Paragraph loops

The renderer enables Docxtemplater's paragraph-loop behavior. Put an opening and closing block tag in their own paragraphs to repeat the paragraphs between them without leaving empty marker paragraphs.

```
{#items}
{name}
{description}
{/items}
```

In the underlying Word XML, a block marker must be inside a text run. A valid marker paragraph contains `<w:t>{#items}</w:t>`. A bare `<w:p>{#items}</w:p>` is not scanned. Word normally creates the correct text run when a tag is typed normally.

Table row loops

To repeat a Word table row, place the opener in the first cell and the closer in the last cell of the row.

First cell	Middle cells	Last cell
`{#items}{name}`	`{amount}`	`{status}{/items}`

Keep the control tags in the row that should repeat. A loop around an entire table should use separate marker paragraphs before and after the table.

Explicit structure expansion

Advanced templates can tell Docxtemplater which Word XML element a block should repeat. The syntax is `{-element expression}`.

```
{-w:tr items}{name}{/items}
{-w:r isApproved}✓{/isApproved}
```

Common Word elements are `w:tr` for a table row, `w:tc` for a table cell, `w:p` for a paragraph, and `w:r` for a formatted text run. Use this only when ordinary paragraph or row loops cannot preserve the required layout. A wrong element or unbalanced block can corrupt the generated document, so inspect the finished file in Word after any change.

The INL templates use `{-w:r condition}` to keep a symbol's formatting while conditionally including the whole run.

Line breaks

The renderer converts newline characters in a value into Office line breaks. This applies to Word and PowerPoint. Keep intentional blank lines in source data; use `collapseWhitespace` when a value should be reduced to one line.

Raw Office XML

Docxtemplater's raw XML tag form, `{@expression}`, is available through the underlying renderer but is not a normal report-authoring tool. It replaces an Office XML container and requires valid XML in the supplied value. Use a supported helper or targeted application code instead unless raw OOXML insertion is explicitly required and tested.

Delimiters and unsupported module tags

The default delimiters are single braces: `{expression}`, not double braces. To show literal brace-heavy content or avoid a delimiter collision, the renderer supports a delimiter-change tag such as `{=<% %>=}`. Subsequent placeholders use `<%expression%>`; `<%={ }=%>` restores braces. Keep delimiter changes rare and test the entire document. The filename is a separate template and does not inherit document delimiter changes.

Image and HTML tags from optional Docxtemplater modules, such as `{%photo}` and `{~html}`, are not part of the general template contract. Use normal Word images for fixed artwork and the documented chart marker for generated indicator charts. HTML strings inserted with a value tag remain text. There is no general include/import tag or automatic `$index` loop variable in this parser.

Word editing hazards

Word can split visually continuous text into several runs after partial formatting, tracked changes, copy/paste, or spell-check corrections. Docxtemplater can join many split tags, but a tag divided across incompatible structures can fail.

When a tag behaves unexpectedly:

1. Retype the whole tag in one action without changing formatting inside it.
2. Put block markers in clean paragraphs or in the intended table row.
3. Accept or reject tracked changes around the tag.
4. Download and generate again before doing XML surgery.

For a targeted XML edit, preserve the file's styles, numbering, theme, fonts, and unrelated package parts.

Missing and false values

An unresolved top-level value tag remains visible in the output:

```
{award.unknownField}
```

This behavior makes missing data loud. Treat a visible tag as an error in the data-source key, property path, or expression.

An absent loop value renders no normal-loop content; an inverse block can supply the fallback. A missing scalar inside a repeated scope can also remain as its original tag, so use explicit fallbacks for optional fields:

```
{description == null ? 'No description provided.' : description}
```

Scalar `0` and `false` render as `0` and `false`. They still count as false in conditional blocks. Use a ternary when you want a human-readable label:

```
{count == 0 ? '0' : count}
{flag ? 'Yes' : 'No'}
```

Data-source values

The configured Key becomes the root name shown in these examples. An editor can choose another key, so inspect the template's Data Sources grid before copying a tag.

Data source	Shape	Typical tags
All Activities	Array of activity objects.	`{#awards}{code}: {title}{/awards}`
All Indicators	Array of indicator objects.	`{#indicators}{code}: {title}{/indicators}`
Single Activity	Activity object with `indicators` and activity metadata.	`{award.title}`, `{#award.indicators}{code}{/}`
Single Activity Reporting Period	Reporting-period object with `award`, `reportingPeriod`, `reportingStatusCode`, `indicators`, and `narratives`.	`{awardReportingPeriod.award.title}`, `{awardReportingPeriod.reportingPeriod.title}`
Single Indicator	Indicator object.	`{indicator.code}`, `{indicator.definition}`
Custom Query	Array of row objects. Query column names are property names.	`{#queryRows}{ProjectName}{/queryRows}`
Data Table	Array of friendly-data row objects. Column labels are property names.	`{#rows}{this['Start Date']}{/rows}`
User Input	String, Boolean, or `null`.	`{notes}`, `{#finalReport}Final{/finalReport}`
Contact	`{displayName, title?, email?}` or `null`.	`{poc == null ? '' : poc.displayName}`

Single Activity Reporting Period

The primary reporting-period object provides the richest standard report context. Its exact fields depend on instance data, but these stable paths are the main authoring surface:

```
{awardReportingPeriod.awardReportingPeriodID}
{awardReportingPeriod.award.title}
{awardReportingPeriod.award.code}
{awardReportingPeriod.award.referenceNumber}
{awardReportingPeriod.reportingPeriod.title}
{awardReportingPeriod.reportingPeriod.startDate}
{awardReportingPeriod.reportingPeriod.endDate}
{awardReportingPeriod.reportingStatusCode.title}
{#awardReportingPeriod.narratives}{Question}: {Answer}{/}
{#awardReportingPeriod.indicators}{code}: {title}{/}
```

Each indicator includes its ordinary DevResults metadata and report-calculation data such as `calculatedProgress`, `currentAggregateResult`, `currentAggregateTarget`, `targets`, `attributes`, `tags`, and `defaultReportingCycle`. Use the shared indicator helpers instead of recomputing results or targets in the template.

Custom Query and Data Table rows

Custom Query returns the query's result array. Data Table returns friendly-data rows. In both cases, author tags against the returned field names exactly.

```
{#rows}
{ProjectName}
{this['Total Budget'] | number:0}
{/rows}
```

Use `findBy` when one row should match a known field exactly:

```
{findBy(rows, 'ProjectName', award.title).TotalBudget | number:0}
```

If the source may not contain a match, guard the result with a condition or ternary.

Contact values

A selected saved contact returns `displayName` and may include `title` and `email`. An explicit Other value returns only `displayName`. No selection returns `null`.

```
{contact == null ? 'No contact selected' : contact.displayName}
{contact == null ? '' : contact.email}
```

Special values

These values are available in every document and generated-filename expression:

Value	Meaning	Example
`_now_`	The live generation date and time.	`{formatDate(_now_, 'YYYY-MM-DD')}`
`_today_`	The local generation date in `YYYY-MM-DD` form.	`{_today_}`
`_host_`	The instance hostname prefixed with `https://` (without a custom port).	`{_host_}`

These values are evaluated when the report is generated. `formatDate` uses UTC date components; the AngularJS `date` filter can use an explicit timezone.

Certification values

Every render receives `certification`. It is `false` unless an eligible user completed the certification flow. A certified value has this shape:

```
{
   "requested": true,
   "completed": true,
   "certified": true,
   "name": "Casey Example",
   "fullName": "Casey Example",
   "email": "casey@example.org",
   "title": "Program manager",
   "date": "2026-08-28"
}
```

Use a block to include a certification section:

```
{#certification}
Certified by {certification.name}, {certification.title}, on {certification.date}
{/certification}
```

The render scope also receives `finalReport`. If a data source with key `finalReport` returns a Boolean, that value wins. Otherwise `finalReport` is true for a certified render and false for an uncertified render.

Shared helper functions

These helpers are available to every report template and generated-filename expression.

`findBy(array, key, value)`

Returns the first array item whose `key` property strictly equals `value`, or `undefined` if none matches.

```
{findBy(award.customFields, 'title', 'Total Project Budget').data}
```

`findBy` does not coerce strings to numbers. The source value and requested value must have the same type.

`truncate(value, maxLength, suffix)`

Converts a non-null value to text. If it is longer than `maxLength`, returns the first `maxLength` characters plus the optional suffix. A short value is unchanged.

```
{truncate(comment, 1000, '...')}
```

`formatDate(value, format)`

Formats a date-like value using this deliberately small token set:

Token	Meaning	Example
`YYYY`	Four-digit year	`2026`
`YY`	Two-digit year	`26`
`MMM`	English abbreviated month	`Aug`
`MM`	Zero-padded month	`08`
`M`	Month without padding	`8`
`DD`	Zero-padded day	`05`
`D`	Day without padding	`5`

```
{formatDate(startDate, 'M/D/YYYY')}
{formatDate(endDate, 'DD-MMM-YY')}
```

An empty or invalid date returns an empty string. Date-only input is parsed without timezone shifting.

`progressBar(percent)`

Creates a native Word progress bar filled to the requested percentage. Values are rounded and clamped to 0–100. A nonnumeric value returns no bar.

```
{progressBar(progressPercent)}
```

Put this tag alone in its own Word paragraph. The renderer replaces that paragraph marker with a small table-based bar.

`collapseWhitespace(value)`

Collapses every whitespace run to one space and trims the result. `null` remains `null`.

```
{collapseWhitespace(definition)}
```

`disaggregationsWithValues(attributes)`

Returns only disaggregations that have a title and at least one titled value. Results are ordered by display order. Values are sorted by title. Each returned item has `title`, `values`, and `valuesText`.

```
{#disaggregationsWithValues(attributes)}
{title}: {valuesText}
{/}
```

`indicatorHasChartData(indicator)`

Returns true when the indicator's calculated report-period progress has an actual or a target.

```
{#indicatorHasChartData(this)}Result or target data is available.{/}
```

`indicatorHasHistory(indicator)`

Returns true when the indicator has at least one plottable historical result or target point.

```
{#indicatorHasHistory(this)}{chartMarker}{/}
{^indicatorHasHistory(this)}No target or result history is available.{/}
```

When chart history exists, DevResults sets the indicator's `chartMarker` and replaces it with a generated PNG during Word rendering. Keep `{chartMarker}` in the intended chart cell or paragraph. If chart rasterization is unavailable, the marker is blank.

Generated-filename syntax

The Generated filename field accepts value tags, expressions, filters, helpers, special values, certification data, and INL helpers. It does not need an extension.

```
Performance report for {awardReportingPeriod.award.code} {awardReportingPeriod.reportingPeriod.title} {_toda
y_}
```

DevResults renders the filename from the same data as the document, removes control characters and `<>:"/\|?*[]`, removes any typed Office or PDF extension, trims trailing periods and spaces, and adds the extension for the selected format.

Use text-returning helpers in filenames. Chart and progress-bar helpers are document layout features and do not produce useful filename text. Every filename expression must resolve. Unlike a missing document value, an unresolved filename expression stops generation with an error.

Syntax and authoring pitfalls

- Do not index a filtered array such as `(rows | filter:x)[0]`; use `findBy` or a helper.
- Keep opening and closing tags balanced. Prefer named closers in long or nested blocks.
- Put paragraph-loop control tags in their own paragraphs.
- Keep a repeating row's opener and closer inside that row.
- Put `progressBar(...)` alone in its own paragraph.
- Keep `{chartMarker}` where the generated image should appear.
- Do not compute indicator results, targets, cumulative periods, weighted averages, or status marks in a template. Use values attached by the production calculation path and the supported helpers.
- Treat a visible unresolved tag as a defect, not acceptable output.

Authoring checklist

Before publishing or replacing a template:

1. Confirm every data source has a unique, stable Key.
2. Check property spelling and case against representative returned data.
3. Generate with populated, empty, and optional-field cases.
4. Exercise every conditional and inverse block.
5. Generate certified and uncertified versions when applicable.
6. Check the generated filename in each supported output format.
7. Open the output in Word, Excel, or PowerPoint and inspect pagination, table repetition, headers, footers, charts, progress bars, and styles.

Appendix: INL-specific helpers

The following helpers exist for INL's report catalog. They encode INL-specific tagging, data-table schemas, custom-query columns, status vocabulary, formatting rules, and report sections. Do not use them in a general template: their names are technically present in the renderer, but their contracts belong to INL data and can change only with the INL templates.

Indicator selection and values

Helper	Return value and purpose
`inlIndicators(indicators, category)`	Printable indicators in `Major Outcome Indicators`, `Minor Outcome Indicators`, or `Activity/Output Indicators`, excluding "do not print" indicators and sorting naturally by code.
`inlOutcomeIndicators(indicators)`	Printable major-outcome indicators used by the INL summary table.
`inlOutcomeTargetCount(indicators)`	Number of printable major-outcome indicators with a current aggregate target.
`inlAllIndicators(indicators)`	All printable indicators sorted naturally by code.
`indicatorResult(indicator)`	INL-formatted calculated result, or `null`. Exact numeric zero is suppressed.
`indicatorTarget(indicator)`	INL-formatted calculated target, or `null`. Exact numeric zero is suppressed.
`indicatorTargetProgress(indicator)`	Whole-percentage calculated progress, or `null` when actual, target, or a nonzero divisor is unavailable.
`aorTargetProgress(indicator)`	The same target-progress convention under the AOR/GOR helper name.

Examples:

```
{#inlIndicators(awardReportingPeriod.indicators, 'Major Outcome Indicators')}
{code}: {title}
Result: {indicatorResult(this) == null ? '-' : indicatorResult(this)}
Target: {indicatorTarget(this) == null ? '-' : indicatorTarget(this)}
Progress: {indicatorTargetProgress(this) == null ? '-' : indicatorTargetProgress(this)}
{/}
```

INL indicator formatting uses the indicator's data format and decimal places. Integers have no decimals, percentages convert a stored ratio to a displayed percentage, and other numbers use fixed precision with locale-aware grouping.

Performance-report shaping

Helper	Return value and purpose
`inlIndicatorComment(indicatorCode, commentRows, awardID, reportingPeriod)`	The scoped CQ54 indicator comment, with INL location prefixes and multirow concatenation, or `null`.
`inlChecklists(rows, award)`	CQ80 rows grouped by checklist heading. Each group has `checkList` and `items`; each item has `done`, `itemName`, `dueDate`, and `comment`.
`inlActivityProgress(rows, award)`	`{planned, completed, percent}` from the INL implementation tracker. Closed-not-applicable rows are not planned; Complete rows are completed; `percent` is null when planned is zero.
`inlTracker(rows, award)`	Award-scoped tracker rows sorted by key. Each row has `code`, `name`, `hasName`, `noItem`, `status`, and `date`.
`inlAdditionalReporting(rows, award)`	Award-scoped Additional Reporting rows with `indicatorName`, `status`, and `comments`.
`inlUpcomingEvents(rows, award)`	Future award events sorted by key, with `name`, `location`, `startDate`, and nullable `leahy`.
`inlSectionNumber(indicators, section, additionalReportingRows?, award?)`	Roman numeral for a conditionally numbered INL section. Supported section names are `tracker`, `additionalReporting`, `upcomingEvents`, `narrative`, `furtherIndicator`, and `certification`.

Example checklist:

```
{#inlChecklists(cq80, awardReportingPeriod.award)}
{checkList}
{#items}{done ? '☒' : '☐'} {itemName} – {dueDate}
{comment}
{/items}
{/}
```

Example optional section:

```
{#inlUpcomingEvents(upcomingEvents, awardReportingPeriod.award).length}
{inlSectionNumber(awardReportingPeriod.indicators, 'upcomingEvents', additionalReporting, awardReportingPeriod.award)}. Upcoming events
{#inlUpcomingEvents(upcomingEvents, awardReportingPeriod.award)}
{name} – {location} – {startDate}
{/}
{/}
```

Indicator Information report shaping

Helper	Return value and purpose
`indicatorMethodCount(indicators, method)`	Count of printable indicators using raw method `entry`, `formula`, or `dynamictable`.
`indicatorDataTableCount(indicators)`	Count of distinct data tables used by printable data-table indicators.
`indicatorDataEntryMethodLabel(method)`	Display label `Direct Entry`, `Formula`, or `Data Table`; unknown non-null values pass through.
`indicatorStandardCount(indicators, award)`	Count of printable indicators whose code does not begin with the activity's code.
`indicatorIndicators(indicators)`	Indicator Information detail population after legacy title exclusions, sorted by code.
`indicatorTargetRows(targets, indicator, today)`	Future target rows after `today`, each with `date` and formatted `targetValue`.
`indicatorGeographies(rows, award)`	Award-scoped CQ301 geography columns combined into rows with `worldRegion`, `country`, `subnational1`, `subnational2`, and `location`.

Example:

```
Direct-entry indicators: {indicatorMethodCount(award.indicators, 'entry')}
Data tables: {indicatorDataTableCount(award.indicators)}

{#indicatorIndicators(award.indicators)}
{code}: {title}
Method: {indicatorDataEntryMethodLabel(dataEntryMethod)}
{#indicatorTargetRows(targets, this, _today_)}
{formatDate(date, 'DD-MMM-YY')}: {targetValue}
{/}
{/}
```

AOR/GOR/COR report shaping

Helper	Return value and purpose
`aorProgressStatusKey(rows, periods, awardReportingPeriod)`	Normalized INL progress-status key for the matching project and period, or `null`. Known keys include `significantlyBelowTarget`, `slightlyBelowTarget`, `onTarget`, `slightlyAboveTarget`, and `significantlyAboveTarget`.
`aorProgressValue(rows, periods, awardReportingPeriod, field)`	Raw field text from the matching progress row, or `null`.
`aorOversightEvents(rows, awardReportingPeriod)`	Oversight events for the project within the selected reporting-period dates, sorted by date. Each item has `type`, `date`, `multiDay`, and `description`.
`aorAllOversightEvents(rows, awardReportingPeriod)`	All oversight events for the project over its lifetime, using the same returned item shape.
`aorFinalEvalText(rows, awardReportingPeriod, field)`	Field text from the project's Final Evaluation data-table row. Missing and whitespace-only values return `null`.
`aorReportLabel(award, pointOfContact)`	`AOR`, `COR`, or `GOR`, based on mechanism type, known INL mechanism fallbacks, and finally point-of-contact text.

Example:

```
{aorReportLabel(awardReportingPeriod.award, poc.displayName)} report

{#aorOversightEvents(oversightLog, awardReportingPeriod)}
{type} {multiDay ? 'starting' : 'on'} {formatDate(date, 'M/D/YYYY')}
{description == null ? 'No description provided.' : description}
{/}
```

The AOR/GOR helpers assume the exact INL data-table and custom-query field labels used by the installed catalog. Use the exact source configuration for the INL catalog; similarly named generic data is not interchangeable.

Troubleshoot an updated template

Symptom	Check
A placeholder remains in the result.	Match the key and field spelling exactly. Confirm the selected record has the field, and provide an explicit fallback for optional values.
A repeated section is missing.	Confirm that the source returns rows and that the loop expression is not false or empty.
Generation reports an unclosed or mismatched tag.	Check both braces and matching block names. Retype the affected tags without tracked changes or partial formatting.
A scope-parser error appears.	Check AngularJS expression syntax. Use object arrays for loops and avoid `{.}`, optional chaining, or JavaScript arrow functions.
The report cannot be generated after changing its filename.	Every filename expression must resolve. Restore the title-based default while checking the expression.
A row repeats with the wrong layout.	Put both loop markers in the same row, or use separate marker paragraphs around a whole table.
A picture is missing.	Check whether this is a fixed Word image or a generated indicator chart. Arbitrary image-module placeholders are not supported.
Users cannot see the revised template.	Check Report Templates access and Visible to groups. Category names do not grant access.

For help, include the template title, selected inputs, output format, and exact error text when contacting **help@devresults.com**.

> Didn't answer your question? Please email us at **help@devresults.com**.

Related Articles