BEVER

Bever Systems
bevercrm.com

BEVER ADVANCED D365 DOCUMENT GENERATION

User Guide

Build a Word template, decide which records may use it, and generate documents from Dynamics 365, without writing code.

Dynamics 365 Online | Dataverse | Microsoft Word



AUDIENCE

For template authors and administrators

EDITION

November 2026

INSIDE THIS DOCUMENT

Contents

1 What it does 4
The two moving parts 4

2 Feature catalogue 5

3 Template syntax reference 7
3.1 Grammar 7
3.2 Field paths 7
3.3 Runtime variables 11
3.4 Literals 11
3.5 Defaults and concatenation 12
3.6 Conditionals 12
3.7 Block markers 13
3.8 Escaping — writing the syntax as text 13
3.9 Line and paragraph breaks inside a placeholder 14

4 Formatting engine 15
4.1 By type 15
4.2 Two-option fields 15
4.3 Examples 16
4.4 Localisation and multi-language 16
4.5 Defaults are configuration 18

5 Repeating and conditional regions 19
5.1 How a region is delimited 19
5.2 Repeating regions 19
5.3 Worked example — a table of a record's related records (1:N) 22
5.4 Conditional regions 23
5.5 Marker placement — worked comparison 23

6 FetchXML engine 24
6.1 Parameters 24
6.2 Supported FetchXML 25
6.3 Paging 25
6.4 Named queries — Document Generation Query 26

7 Availability rules 28
7.1 Model 28
7.2 Operators 28
7.3 How a value is compared 29
7.4 Dotted attribute paths 29
7.5 Semantics 30
7.6 Examples 30
7.7 Limitations 31
7.8 Enforcement 31

8 Setting up templates in Dynamics 365 32
8.1 The template record — Document Generation Template 32

8.2 The Word file is an attachment 32
8.3 When a template is usable 33
8.4 Where a template appears 33
8.5 Adding a template, step by step 33
8.6 Scope — whose access a document reflects 34
8.7 The subscription 34

9 Worked end-to-end example 35
9.1 Configuration 35
9.2 The .docx 36
9.3 What happens 36

10 Troubleshooting 37
10.1 What is deliberately not an error 37
10.2 Common symptoms 37

Appendix — Syntax cheat sheet 40

Everything you need to build a Word template, decide which records may use it, and print it from Dynamics 365 — without writing code.

A template is an ordinary `.docx` file containing `{{…}}` placeholders, attached to a **Document Generation Template** record (`bvr_document_template`). Which records may use it is decided by **Document Generation Availability Rule** records — data, not code. Adding a template, adding a column to a document, changing a date format or offering a template on a new table are all configuration.

> **Who this is for.** Template authors working in Word, and administrators configuring templates, rules and named queries in the Dynamics 365 app. Developers and whoever deploys the solution want `TECHNICAL.md` instead: dependencies, schema constants, custom actions, architecture, internals and the deployment checklist. `README.md` is the map over both.

1 What it does

A user opens a record (or selects several in a grid), clicks **Generate Document** in the ribbon, and picks a template. The framework:

1. asks the server which templates that record is eligible for;
2. renders the chosen `.docx`, replacing placeholders with live Dataverse data;
3. returns the file, which the browser downloads.

Everything the template can express — nested lookups, tables driven by FetchXML, conditionals, formatting — is written **inside the Word document** by a business user. No code change is needed to add a template, add a column to a table, change a date format, or make a template available on a new entity.

The two moving parts

Part	Where it lives	Who edits it
The template — layout and placeholders	A `.docx` attached to a Document Generation Template record	Business user, in Word
The rules — which records may use it	Document Generation Availability Rule records	Administrator, in the app

2 Feature catalogue

#	Feature	Summary
1	**Field substitution**	`{{name}}` → the record's value, formatted in the document's language.
2	**Unlimited nested lookups**	`{{customerid.primarycontactid.parentcustomerid.ownerid.fullname}}`. Each distinct record is fetched once per run.
3	**First related record**	`{{[new_note^new_caseid].subject}}` (1:N), `{{[new_tag^^new_case_tag].name}}` (N:N) — deterministically the oldest.
3a	**First record of a query or relationship**	`{{First(query:Name).new_total}}`, `{{First(relationship:name).new_name}}` — one field off the first row, no region needed.
4	**Runtime variables**	`{{$user.fullname}}`, `{{$now}}`, `{{$today}}`, `{{$recordid}}`, `{{$entityname}}`, `{{$index}}`, `{{$count}}`.
5	**Null defaults**	`{{firstname ?? "Unknown"}}` — chainable. Falls through on **null only**; an empty string is a value.
6	**Concatenation**	`{{firstname + " " + lastname}}` — any expression may be a part.
7	**Inline conditionals**	`If(c1, v1, c2, v2, …, default)` with any number of branches, fully nestable.
8	**Comparison operators**	`=` `<>` `!=` `<` `<=` `>` `>=`, plus `AND` `OR` `NOT` and parentheses.
9	**Emptiness tests**	`field = null` / `field <> null`.
9a	**Safe navigation**	An empty value equals nothing and differs from everything: `field <> "X"` holds when `field` is empty — no manual null guard needed.
10	**Formatting engine**	Per-type defaults + named aliases + .NET format strings, for every Dataverse type.
11	**Repeating regions**	`{{#each …}} … {{/each}}` over table rows or paragraphs; nests to any depth.
12	**Conditional regions**	`{{#if …}} … {{#else}} … {{/if}}` over whole paragraphs or rows.
13	**FetchXML tables**	Full FetchXML in **named query records**: filters, link-entities, joins, aliases, orders, aggregates, grouping, distinct, paging. Never inside the `.docx`.
14	**Query parameters**	Any placeholder may be injected into FetchXML — `{{$recordid}}`, `{{$user.…}}`, a nested lookup, a runtime variable.
15	**Named queries**	Reusable Document Generation Query records, addressed as `{{#each query:Name}}` — shareable between templates and fixable without re-uploading any `.docx`.
16	**Relationship regions**	`{{#each relationship:new_account_cases}}` — 1:N or N:N, no FetchXML needed. Paged, so a region is never quietly truncated.
17	**Enclosing records**	`{{$parent.name}}` reaches out of a region, `{{$parent.$parent.name}}` out of two, `{{$root.name}}` back to the record the document is for.
18	**Headers and footers**	Rendered with the same pipeline as the body.
19	**Multi-record documents**	One file per selection, joined by page break or line break. With page breaks every record keeps its own page setup, headers and footers.
20	**Availability rules**	Attribute / operator / value, no query language. Enforced both in the ribbon and server-side.
21	**Formatting preservation**	Bold labels stay bold; substituted values inherit the placeholder's own run formatting.

#	Feature	Summary
22	**Split-run tolerance**	A placeholder Word split across several runs is still matched — and one split across a *paragraph break* is rejoined before rendering.
23	**Multi-line values**	A value containing newlines becomes real Word line breaks. Characters Word cannot hold are dropped instead of breaking the file.
24	**Localization**	Choice labels, boolean labels and date/number formats follow the template's own language — or, when the template says so, the reader's Dynamics personalization settings.
24a	**Explicit language**	`{{new_type:label:lcid=1049}}` renders that one placeholder in the language you name, whoever is reading.
25	**Caching**	Metadata, records and currencies memoised per run.
26	**Diagnostics**	Structural errors name the placeholder, template and record. Full `ITracingService` output.
27	**Escaping**	`\{{` prints a literal `{{`, so a document can describe the syntax itself.
28	**Scope**	Each template says whether its data is read as system (the default) or as the reader.
29	**Word comments**	Comments in the template are kept, one per record and per repeated row.

3 Template syntax reference

A placeholder is `{{ … }}`, written anywhere text can go: body paragraphs, table cells, text boxes, headers, footers, footnotes and endnotes. Word comments are not rendered: they keep their text as written. They are kept in the document, one per record and per repeated row, and a comment on content an `{{#if}}` removed goes with it. A copied comment shows as a plain comment: only the first keeps its replies.

Text boxes render like the body. A value printed into a text box is never read as template syntax, even if the record holds `{{…}}`.

Footnotes and endnotes read the record their reference mark belongs to. A note whose mark sits in a repeating row reads that row (`{{name}}`, `{{$index}}`, `{{$parent.name}}`), and every row gets its own note with its own values. A note whose mark an `{{#if}}` removed is left out of the document. In a multi-record document each record keeps its own notes, numbered on through the document.

3.1 Grammar

The content of a placeholder is parsed by a recursive-descent parser. Precedence, **loosest first**:

```
coalesce   := or        ( "??" or )*
or         := and       ( "OR"  and )*
and        := not       ( "AND" not )*
not        := "NOT" not | comparison
comparison := concat    ( ( "=" | "<>" | "!=" | "<" | "<=" | ">" | ">=" ) concat )?
concat     := primary   ( "+" primary )*
primary    := "(" coalesce ")"
            | If "(" coalesce ( "," coalesce )* ")"
            | literal
            | fieldPath
```

Because `concat` binds tighter than `comparison`, `a + b = c + d` parses as `(a + b) = (c + d)`.

3.2 Field paths

```
fieldPath  := root? segment ( ( "." | "?." ) segment )*     [ format ]   -- "?.": see below
root       := "$this."                                          -- the record being rendered
            | ( "$parent." )+                                   -- one level out per "$parent"
            | "$root."                                          -- the record the document is for
            | "$" variable "."                                  -- runtime variable, segments optional
            | "First" "(" source ")."                           -- first row of a query or relationship
segment    := attributeName
            | "[" entity "^"  lookupAttribute   "]"     -- first related record via 1:N
            | "[" entity "^^" relationshipName  "]"     -- first related record via N:N
source     := queryName | "query:" queryName | "relationship:" relationshipName
format     := ":" formatSpecifier [ ":lcid=" languageCode ]     -- see §4
            | ":lcid=" languageCode
```

A runtime variable may stand alone — `{{$now}}`, `{{$index}}` — and so may `First(…)`, which then renders that record's primary name. A path rooted on the record must name at least one field.

Root	Meaning
(none)	The record being rendered.

Root	Meaning
`$this.`	Same, written explicitly. Purely for readability.
`$parent.`	The record enclosing the current repeating region. Repeat it — `$parent.$parent.` — to step out of two regions, and so on. **Error** if it reaches further out than there are regions.
`$root.`	The record the document is generated for, from any depth of nesting.
`$name`	A runtime variable.
`First(source)`	The first record of a named query or a relationship.

Every keyword carries the `$` prefix, and no attribute name may begin with one. That is the whole reason the prefix is there: an entity is free to have a column called `parent`, `this` or `root`, and

```
{{$parent.name}}   the enclosing record's name
{{parent.name}}    the name of whatever the attribute called `parent` points at
{{this}}           the attribute called `this`
```

are three different, unambiguous things. A keyword is only ever a keyword at the **start** of a path; `{{ownerid.$parent}}` is an error rather than a surprise.

Examples:

```
{{name}}                                        simple field
{{$this.name}}                                  identical
{{ownerid}}                                     lookup → primary name
{{ownerid.fullname}}                            one hop
{{ownerid.businessunitid.name}}                 two hops
{{ownerid.businessunitid.parentbusinessunitid.name}}   three hops — no practical limit
{{[new_note^new_caseid].subject}}               first note pointing at this case
{{[new_tag^^new_case_tag].name}}                first tag related N:N
{{$parent.ticketnumber}}                        the record enclosing an {{#each}}
{{$parent.$parent.name}}                        two regions out
{{$root.name}}                                  the record the document is for, from any depth
{{First(LastInvoice).new_total}}                first row of a named query
{{First(relationship:new_case_notes).subject}} first related record via a relationship
```

An empty lookup **anywhere** along a path renders as nothing — the whole path short-circuits. An attribute the entity does not define is a **structural error**, not a blank (see TECHNICAL.md §6, `FailOnUnknownAttribute`) — unless the path reaches it through `?.`. **A mistake in a path is an error on every record, whether or not that record has a value there**, so a template that works on a test record cannot fail later on another:

Written	Lookup or column empty	Filled
`{{ownerid.nosuch}}`, no table the lookup can point to defines `nosuch`	**Error**	**Error**
`{{name.x}}`, `name` is a text column	**Error**: *'account.name' is not a lookup*	**Error**
`{{ownerid.fullname.x}}`	**Error**	**Error**

Inside a named query's region the row holds only what the query selects; see §5.2.

Operators `^` and `^^` — which related record is picked

Both markers select **one** record out of potentially many, and the two differ only in how they get there:

Marker	`PathSegmentKind`	Reaches related records through
`[entity^lookupattribute]`	`FirstChildByLookup`	A **1:N** relationship — the child's lookup column pointing back at this record.
`[entity^^relationshipname]`	`FirstChildByRelationship`	An **N:N** relationship — its intersect table.

Given an account with three contacts, `{{[contact^parentcustomerid].fullname}}` prints one of them. Which one?

> **The oldest by `createdon`, with the primary key as a tie-break.**

That is a guarantee, not an accident. The query is issued with an explicit `ORDER BY createdon,` and `TOP 1`. Without an explicit order the platform may return rows in any sequence — index choice, storage layout, and page boundaries all influence it — so the same template over unchanged data could render differently on two consecutive runs. For a printed, archived, possibly legally-significant document that is the worst class of bug: silent, intermittent, and invisible in review.

So specifically, `^` selects:

Candidate rule	Used?
First **created**	✔ **yes** — `createdon` ascending
Tie-break when timestamps are equal	✔ primary key ascending
Newest	✗
First returned by the server	✗ — that is exactly what is being avoided
The "primary" related record	✗ — Dynamics has no such concept for a generic 1:N

When to use them. `^` and `^^` are for the case where business rules guarantee at most one related record — a case's single approval, a contract's single signatory. If more than one can genuinely exist, do not rely on the pick: use a repeating region and show them all, or a named query that filters and sorts to the one you mean.

`First(…)` — one field off a query or relationship

`^` and `^^` reach a single related record through a lookup or an intersect table. `First(…)` does the same job for the two sources a repeating region already understands — a **named query** or a **relationship** — which is what you want as soon as the record you mean needs filtering or sorting to identify, or is simply easier to name than to navigate to.

```
{{First(<source>).<path>}}

{{First(LastInvoice).new_total}}                    bare name = a named query
{{First(query:LastInvoice).new_total}}              the same, written explicitly
{{First(relationship:new_case_notes).subject}}      through a relationship
{{First(relationship:new_case_notes)}}              the record's primary name
{{First(LastInvoice).ownerid}}                      a lookup on the row: its name
{{First(relationship:new_case_notes).ownerid.fullname}}   a relationship's row keeps traversing, like any path
{{First(LastInvoice).new_status:label:lcid=1033}}   formats and languages apply as usual
```

It is an ordinary field path, so it works everywhere one does: in `{{#if}}`, inside `If(…)`, either side of `??`, as part of a concatenation, and as a FetchXML parameter. A **query's** first row holds only what its FetchXML selects, exactly as in a region (§5.2): read a linked table through a `link-entity` alias, `{{First(LastInvoice).Owner.fullname}}`, not through the lookup.

Which row is "first".

Source	The row you get
`query:`	The **query's own first row** — whatever its `` says. The query is asked for one row (`top="1"`) where its shape allows.
`relationship:`	The **oldest** by `createdon`, primary key as a tie-break — the same guarantee `^` and `^^` give.

So a named query is where you decide "first" for yourself: order by `new_date` descending and `{{First(Invoices).new_total}}` is the *latest* invoice. A query with no `` has no defined first row — add one.

An empty source is empty, not an error. No rows means the placeholder renders as nothing, exactly like an unset lookup, and `{{First(Q).field ?? "—"}}` fills in the dash.

Cost. The first row of a source is fetched **once per run** and reused, so a header block reading five fields off `{{First(LastInvoice)…}}` costs one retrieve, not five. A query whose parameters differ per row — one filtered by `{{$recordid}}` inside a region — is naturally fetched once per row, because its parameters are what the caching is keyed on.

Optional attributes — `?.`

A polymorphic lookup — a Customer column such as `customerid`, an Owner, a Regarding — points at records of more than one table, and the tables rarely share every column. Writing `?.` instead of `.` makes the **next** segment optional: when the referenced record's table does not define that attribute, the path is empty instead of an error. Combined with `??`, one placeholder picks whichever attribute the record has:

```
{{ownerid?.internalemailaddress ?? ownerid?.emailaddress}}
```

`ownerid` points at	Renders
a user (`systemuser` defines `internalemailaddress`)	the user's `internalemailaddress`
a team (`team` defines `emailaddress`)	the team's `emailaddress`
nothing	*(empty)*

The same shape gives a user's or a team's name, `{{ownerid?.fullname ?? ownerid?.name}}`, and a contact's or an account's, `{{customerid?.fullname ?? customerid?.name}}`.

Written	`ownerid` is empty	`ownerid` is a user
`{{ownerid.fullname}}`	*(empty)*	the name
`{{ownerid.emailaddress}}`	*(empty)*: `team` defines it	**Error**: `systemuser` does not
`{{ownerid.nosuch}}`	**Error**: no target defines it	**Error**
`{{ownerid?.nosuch}}`	*(empty)*	*(empty)*

The rules:

- `?.` tolerates **only** an attribute the table does not define. Everything else is unchanged: a typo in the attribute *before* the `?.` is still an error, and following a text column as though it were a lookup is still an error, on every record. On a named query's row, reading through a lookup is an error even with `?.`.
- It covers **one** segment. In `{{customerid?.parentcustomerid.name}}` the `parentcustomerid` is optional, but `name` on whatever it points at is still required. Write `?.` again for each segment that may be missing.
- `.` stays strict: `{{ownerid.emailaddress}}` still stops the run on a user-owned record. Use `?.` where a missing column is expected, and keep `.` where it would be a mistake.
- It works everywhere a path does — output, `??`, `If(…)`, `{{#if}}`, comparisons, FetchXML parameters — and with formats: `{{customerid?.new_businessname:upper}}`. When you want to branch on the table rather than on a column, compare its entity type: `{{If(customerid:entitytype = "contact", …, …)}}` (see §4.3).

3.3 Runtime variables

Variable	Type	Notes
`{{$user}}`	EntityReference	The user the document is generated for.
`{{$user.fullname}}`	any	Traverses into the `systemuser` record; further hops allowed.
`{{$now}}`	DateTime	One timestamp for the whole run, so it is identical everywhere in a document. **Printed in the time zone of whoever generates the document** (their personalization setting), exactly like a *User Local* date/time field such as `createdon`. Conditions compare it the way it prints (§4.4), and it is passed to FetchXML as UTC (§6.1).
`{{$today}}`	Date	**The generating user's calendar date** — midnight in their time zone, not the UTC date. Renders as a date (`dd.MM.yyyy`) unless the placeholder names a format, and is never converted again. In FetchXML it is written as `yyyy-MM-dd` (§6.1).
`{{$recordid}}`	Guid	Current scope's record id — the usual FetchXML parameter.
`{{$entityname}}`	string	Current scope's entity logical name.
`{{$index}}`	int	1-based position in the enclosing repeating region. `1` outside one.
`{{$count}}`	int	Row count of the enclosing region. `1` outside one.

`$now` and `$today` fall back to UTC when the user's personalization cannot be read (§4.4) — generation is not refused.

3.4 Literals

Kind	Syntax	Notes
String	`"text"`	Escapes: `\"` `\\` `\n` `\r` `\t`.
Number	`42` , `-5` , `1.5`	Invariant decimal point.
Boolean	`true` / `false`	Case-insensitive.
Null	`null`	Case-insensitive.

Kind	Syntax	Notes
Guid	`3f2504e0-4f89-11d3-9a0c-0305e82c3301`	Bare, unquoted.
Date/time	`"2026-01-31T00:00:00Z"`	**Must be quoted** — an unquoted colon starts a format specifier.

3.5 Defaults and concatenation

```
{{firstname ?? "Unknown"}}
{{ownerid.fullname ?? parentaccountid.ownerid.fullname ?? "Unassigned"}}
{{firstname + " " + lastname}}
{{"Case " + ticketnumber + " (" + statuscode + ")"}}
```

`??` falls through on **null only** — an attribute with no value at all, or anything reached through an empty lookup. Everything else is a value and wins, including an empty string:

`firstname`	`{{firstname ?? "Unknown"}}`
never set	`Unknown`
`""`	*(empty)*
`"Jane"`	`Jane`

The decision is made on the **evaluated value**, not on the text it renders to, so a part that holds something and happens to print as nothing still wins — an empty text column, a choice whose label is blank, a format that yields `""`. A template prints what the record says.

That also means the parts of a `??` chain that can never be null never fall through: a concatenation always produces a string, so `{{firstname + " " + lastname ?? "Unknown"}}` renders a lone space rather than `Unknown`. Coalesce the fields, not the expression: `{{(firstname ?? "Unknown") + " " + lastname}}`.

When you *do* want blank text treated as missing, use `If` — `=` deliberately reads an empty string as empty, so one condition covers both:

```
{{If(firstname = null, "Unknown", firstname)}}
```

3.6 Conditionals

`If` takes any number of `condition, value` pairs followed by a mandatory default. The argument count is therefore always odd and at least three.

```
{{If(new_batch = null, "", "Start Time")}}
{{If(new_batch = null, "", new_begin_time:time)}}
{{If(new_days_open > 30, "Critical", new_days_open > 15, "High", new_days_open > 5, "Medium", "Low")}}
{{If(new_text <> null AND new_count > 10, "Red", "Green")}}
{{If(new_text = null OR new_count <= 10, "Green", "Red")}}
{{If(NOT new_isclosed, "Open", "Closed")}}
{{If(statuscode = 1, "New " + name, statuscode = 2, "Active " + name, name)}}
{{If(a = 1, If(b = 1, "A1B1", "A1"), "other")}}
```

Empty values in comparisons — no manual null checks

A comparison written against a path **guards itself**. An empty value — an unset attribute, an empty text field, or anything reached through an empty lookup — is **equal to nothing and different from everything**:

Written	`ownerid` empty	`ownerid` = Jane	`ownerid` = someone else
`ownerid.fullname = "Jane Doe"`	not satisfied	satisfied	not satisfied
`ownerid.fullname <> "Jane Doe"`	**satisfied**	not satisfied	satisfied

So `{{If(new_status <> "Closed", "open-ish", "closed")}}` already covers the record whose status was never set — there is no need to write `new_status = null OR new_status <> "Closed"`.

Ordering an empty value is meaningless, so `<`, `<=`, `>` and `>=` stay unsatisfied: an unassigned record never "scores above ten".

Writing `null` explicitly is still the way to test emptiness directly:

```
{{If(new_batch = null, "not started", "started")}}
{{If(ownerid <> null, "assigned", "unassigned")}}
```

`AND` / `OR` short-circuit, so an explicit guard is never wrong — it is simply not necessary:

```
{{If(ownerid.fullname = "Jane Doe", "Yes", "No")}}
{{If(ownerid <> null AND ownerid.fullname = "Jane Doe", "Yes", "No")}}
```

Type coercion

Comparisons coerce by value type, so a choice can be matched either way:

```
{{If(statuscode = 1, "Open", "Closed")}}          by value
{{If(statuscode = "Open", "Open", "Closed")}}     by localized label
```

Coercion is attempted in this order: **Guid** → **Boolean** → **DateTime** → **option-set label** → **number** → **text** (case-insensitive). A lookup compares equal to a bare GUID *or* to its display name.

3.7 Block markers

Marker	Meaning
`{{#each }}` … `{{/each}}`	Repeating region.
`{{#if }}` … `{{#else}}` … `{{/if}}`	Conditional region. `{{#else}}` optional.

Full details in §5.

3.8 Escaping — writing the syntax as text

Put a backslash before an opening delimiter to make it literal:

Written in Word	Renders as
`\{{`	`{{`

Written in Word	Renders as
`\{{name}}`	`{{name}}`
`\{{#each Items}}`	`{{#each Items}}`
`\{{#if x}}`	`{{#if x}}`
`}}`	`}}`

One escape at the **start** is enough for a whole marker, because a closing `}}` on its own is never interpreted — the scanner only ever looks for an opening delimiter. That is why there is no `\}}`: it would be a second rule to learn that buys nothing, and it would quietly eat a backslash from anyone whose prose legitimately ends with one.

This matters for documents that describe the template language itself — a user guide printed from Dynamics 365, for instance — and it is the only way to get a literal `{{` into output.

3.9 Line and paragraph breaks inside a placeholder

A placeholder does not have to sit on one line. Word splits what you type into runs as you go — a spell-check mark or a language change is enough — and that has never mattered: a placeholder is matched against the paragraph's text, not run by run. Pressing **Enter** in the middle of one is also fine:

```
Dear {{If(new_gender = 1, "Mr", "Ms") + " " +
lastname}}, thank you for your order.
```

renders as one continuous sentence. The paragraph break inside the placeholder is removed with it, because you were writing one paragraph and the break was only there to keep the line readable in Word.

Two things to know:

- The break is read as either **nothing** or **a space**, whichever makes the placeholder valid. So both `{{first` / `name}}` and `{{#each` / `query:Items}}` work.
- If the braces do not close into something the engine can read, nothing is joined. That is deliberate: it is what keeps an unescaped `{{` in ordinary prose — see §3.8 — from swallowing the paragraph after it.

4 Formatting engine

A format specifier follows the field after a colon, and an optional `lcid=` qualifier follows the format: `{{field:format}}`, `{{field:format:lcid=1033}}` — see Pinning a language on one placeholder.

> **Quoting rule.** An unquoted specifier may not contain a space, a comma, or a bracket. Quote it when it does: `{{createdon:"MMMM dd, yyyy"}}`, `{{new_channels:" / "}}`. Colons and slashes *are* allowed unquoted, so `{{t:HH:mm}}` and `{{d:dd/MM/yyyy}}` work as written.

4.1 By type

Dataverse type	Default	Aliases	Also accepts
String / Memo	as stored	`upper`, `lower`, `trim`	—
Two options	the field's own labels	`label`, `text`, `checkbox`, `crosscheckbox`, `checkmark`	any `A/B` pair
DateTime	`dd.MM.yyyy HH:mm` — date-only fields `dd.MM.yyyy`	`date`, `time`, `datetime`	any .NET date format
Integer / BigInt	`N0`	`percent`	any .NET numeric format
Decimal / Double	`N` *precision*	`percent`	any .NET numeric format
Money	`N` *precision*	`currency` (symbol), `code` (ISO code), `percent`	any .NET numeric format
Choice / State / Status	localized label	`value`, `label`	—
Multi-choice	labels joined by comma + space	`value`	any string, used as the separator
Lookup / Customer / Owner	primary name	`name`, `id`, `entitytype`	—
Unique identifier	`D`	—	any GUID format

4.2 Two-option fields

Five named renderings, all interchangeable on any two-option field:

Specifier	`true`	`false`
(none) / `label`	the field's own **Yes** label	the field's own **No** label
`text`	`Yes`	`No`
`checkbox`	☑	☐
`crosscheckbox`	☑	☒
`checkmark`	✓	✗

```
{{approved}}                  → Yes          (the field's label)
{{approved:label}}            → Yes          (identical — label is the default)
{{approved:text}}             → Yes
{{approved:checkbox}}         → ☑  /  ☐
{{approved:crosscheckbox}}    → ☑  /  ☒
{{approved:checkmark}}        → ✓  /  ✗
{{approved:Agreed/Declined}}  → Agreed       (any explicit pair)
```

`label` **is the default** because the field's own labels are what the user sees on the form, what an administrator can rename, and what Dynamics already translates. Rename a field's options to *Agreed / Declined* and every document that prints it follows, with no template edited. Use `text` when you deliberately want language-neutral words regardless of the field's labels.

> **Font note.** ☑ ☐ ☒ ✓ ✗ are ordinary characters, not form controls, so they carry no interactive state and need a font that has them — Segoe UI Symbol, DejaVu Sans and most Word default fonts do.

4.3 Examples

```
{{new_date:HH:mm}}                → 09:30
{{new_date:dd/MM/yyyy}}           → 15/03/2024
{{new_date:date}}                 → 15.03.2024
{{new_date:"MMMM dd, yyyy"}}      → March 15, 2024
{{createdon:yyyy-MM-dd}}          → 2024-03-15

{{revenue}}                       → 1,234.50
{{revenue:N0}}                    → 1,235
{{revenue:currency}}              → 1,234.50 $
{{revenue:code}}                  → 1,234.50 USD
{{new_ratio:percent}}             → 12.50 %

{{statuscode}}                    → Active
{{statuscode:value}}              → 1
{{new_channels}}                  → Email, Phone
{{new_channels:" / "}}            → Email / Phone
{{new_channels:value}}            → 1, 2

{{ownerid}}                       → Jane Doe
{{ownerid:id}}                    → 2222…-2222
{{ownerid:entitytype}}            → systemuser
```

`:entitytype` is the logical name of the table the lookup points at — not the lookup column's own name. It is the same value in a condition as in output, so `{{#if ownerid:entitytype = "team"}}` and `{{If(customerid:entitytype = "contact", …, …)}}` compare the table name, never the record's name.

4.4 Localisation and multi-language

A document renders in **one language**, decided by the template record:

Use Current User Selected Language	*Language*	The document renders in
Yes	*(ignored)*	the language of whoever generates it — their Dynamics personalization setting

Use Current User Selected Language	*Language*	The document renders in
No	set, e.g. Russian (1049)	that language, whoever generates it
No	empty	the language of whoever generates it

Set *Language* to the language the template is **written in**, so the parts that come from data match the prose around them. Use *Use Current User Selected Language* for templates whose prose is neutral (a table, a list of fields) and that each user should get in their own language.

The document language decides:

What	Effect
Choice labels — including two-option `label` renderings	The option's label in the document language, falling back to the default label
Numbers, currency, dates	The culture of the document language: month and day names, decimal and grouping separators

The template **name** is shown as written, in every language.

- **Time zone.** A date column prints what the same user sees on its form, decided by the column's **Behavior**, and shows a time unless its **Format** is *Date Only*. For a column holding `2026-08-02 20:30`, a user in UTC+4 gets:

Behavior	Format	Converted to the user's time zone	Prints
User Local	Date and Time	Yes	`03.08.2026 00:30`
User Local	Date Only	Yes	`03.08.2026`
Time Zone Independent	Date and Time	**No**	`02.08.2026 20:30`
Time Zone Independent	Date Only	**No**	`02.08.2026`
Date Only	—	**No**	`02.08.2026`

  An explicit format changes only the layout: `{{tzi_column:HH:mm}}` prints `20:30` for everyone. Aliased columns of a named query follow the rules of the column they alias.
- `{{$now}}` prints like a User Local column, with any format and any `lcid=` (`{{$now:"dd.MM.yyyy HH:mm"}}` prints the user's wall-clock time), and `{{$today}}` is the user's own date — between 00:00 and 04:00 in UTC+4 it is already today, not yesterday.
- **Conditions compare what prints.** `{{If(checkin = "2026-08-02 20:30", …)}}` holds for the Time Zone Independent column above, `{{If(createdon >= $today, …)}}` means "since midnight for this user", and text with no zone of its own means that time for this user. Text that names one, like `2026-08-02T20:30:00Z`, means exactly that moment.
- If the user's personalisation is unavailable, the document renders in **UTC** (and, when it follows the user, with the default labels) — a warning is traced, generation is not refused.
- The language must be **enabled in the environment** for its labels to exist; otherwise the default labels are used.

The template *body* is not translated: one document, in whatever language its author wrote it. Translating prose means a template per language — each with its own *Language* — related by whatever the availability rules choose to match on. What is translated is everything that comes **from data**, which is where duplication would actually hurt.

Pinning a language on one placeholder

Everything above follows the document language. When one line has to be in a **different** language — a bilingual contract, a form for a foreign counterparty, a line that is quoted verbatim from a regulation — name the language after the format:

```
{{new_type:label:lcid=1049}}          the choice's Russian label, whoever is reading
{{new_type:label:lcid=1033}}          the same choice, in English
{{new_type:lcid=1049}}                no format, Russian label
{{createdon:"d MMMM yyyy":lcid=1049}} 9 марта 2026
{{createdon:"d MMMM yyyy":lcid=1033}} 9 March 2026
{{revenue:N2:lcid=1049}}              1 234,50   (Russian digit grouping and decimal comma)
```

The number is a **Dynamics language code (LCID)** — 1033 English, 1049 Russian, 1067 Armenian, and so on: the same codes the *Languages* page of the environment settings lists.

`lcid=` **is always written.** A bare `{{new_type:label:1049}}` is *not* a language: it is a format specifier that happens to read `label:1049`. That is deliberate — a format may legitimately end in a number, so requiring the prefix is what guarantees no placeholder written before this existed means something different now.

Where it applies	Effect
Choice, multi-choice, state, status	The option's label in that language.
Two options rendered as `label`	The field's own Yes/No label in that language.
Dates, numbers, money	That language's culture: month and day names, decimal and grouping separators.
`upper` / `lower`	Cased by that language's rules rather than invariantly.
Lookups, text, GUIDs	Nothing — these come from data, and data is not translated.

Two positions, one spelling:

Written	Reads as
`{{field:format:lcid=1049}}`	A format, then the language it renders in.
`{{field:lcid=1049}}`	A language with **no** format — the type's default rendering, in that language.

Nothing else changes meaning: `{{d:HH:mm}}` is still minutes, `{{d:HH:00}}` still prints a literal `00`, and `{{n:1049}}` is still a numeric format. A specifier that starts `lcid=` but names no valid code — `{{c:label:lcid=ru}}` — is a template error rather than a silently ignored format, because it can only be a typo.

If the language is missing. An option with no translation into the language you named falls back to the document language, and then to its default label — a missing translation shows the untranslated text rather than a blank line, which is far easier to spot and to fix.

4.5 Defaults are configuration

Every default above lives in `Configuration/FormattingDefaults.cs`. Changing the house date format is a one-line edit there, not a change to the formatting engine.

5 Repeating and conditional regions

5.1 How a region is delimited

A region is bounded by two markers written **anywhere** in the document. The processor raises them to the closest level at which they are siblings and repeats or keeps that range:

Where the markers are	What repeats
Two cells of **one table row**	that row
Cells of **two different rows**	the rows between them, inclusive
Two **paragraphs**	those paragraphs and everything between them, inclusive
Both in **one paragraph**	that paragraph

Two markers in different cells of the same row are promoted to the **row**, never the cells — a repeated cell would corrupt the table geometry.

A marker alone on its own line is removed **without leaving a blank line**: empty boundary paragraphs are trimmed from the region.

A region never removes layout from the marker's own line. A section break, page break or bookmark on the same line as a marker is not part of the region: it stays exactly once, whether the condition holds or not and however many rows there are, and an empty line remains to hold it. To repeat a page break per row, or to make one conditional, put it in a paragraph of its own **between** the markers:

```
{{#each query:Q}}
{{name}}
<page break>        repeats per row
{{/each}}<page break>        stays once, after the last row
```

Everything between the markers belongs to the region, so a false `{{#if}}` removes what is inside it, including any section break there.

5.2 Repeating regions

```
{{#each <source>}} … {{/each}}
```

Source syntax	Meaning
`{{#each Contacts}}`	Named query record (bare name is the shorthand).
`{{#each query:Contacts}}`	Same, written explicitly.
`{{#each relationship:contact_customer_accounts}}`	Records related through a 1:N or N:N relationship.

Prefer `relationship:`. It needs nothing but the relationship's schema name, has no query to maintain anywhere, and covers the overwhelmingly common case of "list this record's children". Reach for a named query only when you need filtering, sorting, aggregation, or columns from further out.

There is no way to write a query inside the Word file.

One record rather than a list? Use `{{First().}}`. It takes the same two sources and reads a single field off the first row, with no region to delimit.

A repeating table row, driven by a relationship:

Ticket	Opened	Account
`{{#each relationship:incident_customer_accounts}} {{ticketnumber}}`	`{{createdon:date}}`	`{{$parent.name}}{{/each}}`

A repeating paragraph block:

```
{{#each query:OpenCases}}
Case {{$index}} of {{$count}}: {{ticketnumber}}
Opened {{createdon:date}} for {{$parent.name}}
{{/each}}
```

Inside a region:

- `{{field}}` addresses the **row**;
- `{{$parent.field}}` addresses the **record that contains the region**, `{{$root.field}}` the record the document is for;
- `{{$index}}` / `{{$count}}` are the loop counters;
- FetchXML **aliases** are readable as ordinary fields, with the rules below.

A named query's row holds only what its FetchXML returns. Every column the template reads from a row must be selected by the query, and a linked table's columns come through a `link-entity` with an alias:

Inside `{{#each query:Cases}}`	Result
`{{bvr_name}}`, selected by the query	The value
`{{bvr_notes}}`, **not** selected by the query	Error: *'query:Cases' does not select the column 'bvr_notes'*
`{{ownerid}}`, `{{ownerid:id}}`, `{{ownerid:entitytype}}`	The owner's name, id or table, straight from the row
`{{ownerid.fullname}}`, `{{[contact^parentcustomerid].fullname}}`, with or without `?.`	Error: add a `link-entity` to the query instead, and read `{{User.fullname}}`
`{{$parent.…}}`, `{{$root.…}}`	Unchanged: they read their own records

This keeps a region to one request per page of rows, and it means a forgotten column is an error rather than a blank. A relationship region (`relationship:…`) is not FetchXML, and reads through lookups as usual.

Aliases and linked columns of a named query. A value is formatted by the column it came from, so an aliased choice still renders as its label and an aliased date follows §4.4.

```
<fetch>
  <entity name="bvr_testcase">
    <attribute name="bvr_name" />
    <attribute name="bvr_os_status" alias="CaseStatus" />
    <link-entity name="systemuser" from="systemuserid" to="owninguser" link-type="outer" alias="User">
      <attribute name="fullname" />
      <attribute name="domainname" alias="Email" />
      <link-entity name="businessunit" from="businessunitid" to="businessunitid" link-type="outer" alias="BusinessUnit">
        <attribute name="name" />
      </link-entity>
    </link-entity>
  </entity>
</fetch>
```

The query says	Write
`alias` on an `attribute`, of the query's table or a linked one	`{{CaseStatus}}`, `{{Email}}` — the alias alone, not `{{User.domainname}}`
`alias` on a `link-entity`, attribute with none	`{{User.fullname}}`
A `link-entity` inside another	`{{BusinessUnit.name}}` — its own alias, not `{{User.BusinessUnit.name}}`
An aliased lookup, `>`	`{{Owner}}` for its name. To read further, link its table: `{{Owner.fullname}}` is an error on a query row

These work wherever a row is read: `{{First(query:Cases).User.fullname}}`, `{{$parent.Email}}`, `{{#if}}`, `If(…)` and `??`. Aliases are case sensitive.

A `link-entity` alias follows the same rules as a lookup on a record, with `User` in the place of `ownerid`:

Written	The join found nothing, or the value is null
`{{User.fullname}}`	Empty
`{{User?.fullname}}`	Empty
`{{User.nosuch}}`	Error, on every row: *The query's link-entity 'User' does not select 'nosuch'*
`{{User?.nosuch}}`	Empty

An alias the query does not declare at all, `{{Nope.x}}` or `{{Nope?.x}}`, is always an error, as an unknown first column is.

Money is printed in the currency of the table it came from, including through an alias or a `link-entity`: the engine adds `transactioncurrencyid` to the query itself. The exceptions are `aggregate` and `distinct` queries, where an extra column would break or change the result. A `distinct` query that selects `transactioncurrencyid` itself gets its symbol.

Give every `link-entity` that selects columns an `alias`. Without one Dataverse invents a name (`systemuser1.fullname`) the template cannot rely on, so such a query is refused: "*the link-entity 'systemuser' selects columns but has no alias. Give it an alias.*" A link that only filters needs none, and neither does one whose columns each have their own alias. A `link-type="matchfirstrowusingcrossapply"` link returns its columns under their **schema names** instead, `{{FullName}}`, or under their own alias, and they are empty on a row with no match.

Regions **nest to any depth**, and an inner region sees the outer row as its `$parent`. Each `$parent` steps out of exactly one region, so an inner region reaches the record the document is for with `{{$parent.$parent.field}}` — or, without having to count, with `{{$root.field}}`:

```
{{#each relationship:new_account_orders}}
Order {{name}}
  {{#each relationship:new_order_orderlines}}
  - {{new_product}} for order {{$parent.name}} ({{$root.name}})
  {{/each}}
{{/each}}
```

The inner region resolves its relationship against the **order**, because it is evaluated in the row scope — and `{{$parent.name}}` inside it reaches back to that order. A named query used in the same position would see `{{$recordid}}` as the order's id for the same reason.

An empty result set removes the region entirely.

5.3 Worked example — a table of a record's related records (1:N)

Goal. On an `account`, print every related `contact` as a table:

Name	Email
John	john@test.com
Jane	jane@test.com

In Word, build a two-column table with a header row and **one** body row. The body row is the template for every contact; put the opening marker at the start of its first cell and the closing marker at the end of its last:

Name	Email
`{{#each relationship:contact_customer_accounts}}{{fullname}}`	`{{emailaddress1}}{{/each}}`

That is the whole template. What happens:

1. Both markers sit in the same table row, so the processor raises them to that **row** and repeats it.
2. The relationship is resolved from metadata; because `contact_customer_accounts` is 1:N, the rows are the contacts whose `parentcustomerid` points at this account.
3. The template row is cloned once per contact, the markers are removed, and each clone is filled against **its own** contact — `{{fullname}}` and `{{emailaddress1}}` address the row, not the account.
4. The header row is untouched; it contains no markers.
5. No contacts → the row disappears and only the header remains.

Variations.

```
{{$parent.name}}                    the account, from inside the row
{{$index}} of {{$count}}            row number and total
{{createdon:dd/MM/yyyy}}            formatted per column, as anywhere else
{{#each relationship:…}}…{{/each}}  a second table, for a different relationship
```

Spanning more than one row. Put the markers in the **first and last** row of the block you want repeated — every row between them repeats together:

`{{#each relationship:contact_customer_accounts}}{{fullname}}`	`{{emailaddress1}}`
`{{address1_city}}`	`{{telephone1}}{{/each}}`

N:N works identically — `{{#each relationship:new_account_tag}}` — because the processor asks metadata for the relationship's cardinality and builds the right query itself.

When to use a named query instead. Only when the plain child list is not what you want: filtering (`statecode = 0`), sorting, aggregation, paging a very large set, or columns from a further-out entity. Then create a Document Generation Query and write `{{#each query:OpenCases}}` — the Word side is unchanged.

How related records are loaded. One query per region, executed once per enclosing row, paged automatically, capped by `MaxRegionRows`. Rows are read in the run's scope, so a contact the user cannot see simply does not appear.

5.4 Conditional regions

```
{{#if new_amount > 1000000}}
Requires board approval by the Credit Committee.
{{#else}}
Standard approval applies.
{{/if}}
```

- `{{#else}}` is optional and **must start its own paragraph or table row**.
- The else element belongs to the false branch: text following the marker on that line is false-branch content.
- For conditions **inside a line**, use the inline `If(…)` function instead. Attempting to put `{{#else}}` in the same paragraph as `{{#if}}` raises an explicit error saying so, rather than silently rendering the wrong branch.
- `{{#if}}` regions may contain `{{#each}}` regions and vice versa.

5.5 Marker placement — worked comparison

```
✓  | {{#each query:Rows}}{{name}} | {{amount}}{{/each}} |          repeats the one row

✓  | {{#each query:Rows}}{{name}} | {{amount}}          |
   | {{description}}              |          {{/each}}  |          repeats both rows

✗  {{#if x}}yes{{#else}}no{{/if}}                                   → error: use If(x, "yes", "no")

✓  {{If(x, "yes", "no")}}                                           inline conditional
```

6 FetchXML engine

FetchXML survives in exactly **two** places, both of them outside the Word file:

Where	Who writes it
Document Generation Query records — `{{#each query:Name}}`	An administrator, in the app
(nothing else)	—

Neither availability rules nor relationship regions involve FetchXML: a rule is a single comparison against an attribute, and a relationship region is built from the relationship's own metadata. A template author never sees FetchXML at all unless they choose a named query, and even then they only write its name.

6.1 Parameters

Any placeholder may be embedded in a named query's FetchXML and is evaluated in the **current scope**:

```
<fetch>
  <entity name="incident">
    <attribute name="ticketnumber" />
    <attribute name="createdon" />
    <filter type="and">
      <condition attribute="customerid" operator="eq" value="{{$recordid}}" />
      <condition attribute="ownerid"    operator="eq" value="{{$user:id}}" />
      <condition attribute="statecode"  operator="eq" value="0" />
      <condition attribute="createdon"  operator="on-or-after" value="{{new_startdate}}" />
    </filter>
    <order attribute="createdon" descending="true" />
  </entity>
</fetch>
```

Values are written in **invariant form**, not the document's language, so they compare correctly:

Value type	Written as
Guid / EntityReference	`D` format GUID
DateTime — User Local fields, `{{$now}}`	`yyyy-MM-ddTHH:mm:ssZ` (UTC)
DateTime — Time Zone Independent fields	`yyyy-MM-ddTHH:mm:ss`, as stored, no zone
Date — Date Only fields, Time Zone Independent + Date Only fields	`yyyy-MM-dd`, as stored, no time
`{{$today}}`	`yyyy-MM-dd` — the generating user's date, no time
Choice	numeric value
Boolean	`1` / `0`
Money / numeric	invariant decimal
Text	as-is
Null	empty string

Dates in conditions. `{{$today}}` is written without a time so Dataverse treats it as a whole day:

- against a **date-only** column (`on`, `on-or-after`, `on-or-before`, `eq`, …) it compares exactly that calendar date — no shift;
- against a **date/time** column, `on` / `on-or-after` / `on-or-before` take the day's boundaries in the time zone of the user **the query runs as** — the reader for a template whose Scope is User, the SYSTEM account for one whose Scope is System. When the exact moment matters, compare against `{{$now}}`, which is always UTC.

Every substituted value is **XML-escaped**, and the result is parsed before execution — a value containing a quote or an angle bracket cannot restructure the query. Substitution runs **exactly once**, so a `{{` that arrives as part of record data is never re-interpreted. A character XML cannot hold, such as one pasted from another program, is dropped from the value first.

6.2 Supported FetchXML

The engine does not restrict FetchXML — filters, nested `link-entity`, inner and outer joins, `alias`, `order`, `aggregate`, `groupby`, `distinct` are all passed through, with two rules:

- a `link-entity` that selects columns must have an `alias` (§5.2);
- the template may read only what the query selects (§5.2).

6.3 Paging

A query with no limit of its own returns **every** matching row: the engine pages through the results until they are exhausted or the row limit is reached.

A query that limits itself runs **once, as written**, for `{{#each}}` and `{{First(…)}}` alike:

	Returns
`top="2"`	The first 2 rows
`count="2"`	The first 2 rows — the first page, the same as `top="2"`
`count="2" page="2"`	2 rows from the second page: rows 3 and 4
`page="2"`	The second page at the default page size of 5000
`aggregate="true"`, `distinct="true"`	The single result set as written, at most 5000 rows: more is an **error** naming the query, never a silently shorter table. Narrow the filter or add `top`.

This holds wherever a named query is used, because they all run the same way: a region, a nested region, `{{First(…)}}` inside `If(…)`, `??` or `{{#if}}`, another query's parameter, and an availability rule's path.

`{{$count}}` counts the rows returned, so `count="2" page="2"` gives `2` even when more rows match, and `{{First(…)}}` is the first row of that page. Dataverse limits this kind of paging to the first 50,000 rows of a query.

Order. When a query with no limit of its own is sorted, the engine adds the primary key as the last sort, so rows that share a sort value always come out in the same order and paging never repeats or skips one. It does not when an sits inside a `link-entity`, because the linked sort would then stop applying: end such a query's sort on a unique column yourself. A query you page with `count` and `page` should do the same.

6.4 Named queries — Document Generation Query

A named query is a **record**, not a field. `{{#each query:OpenCases}}` looks for an active Document Generation Query (`bvr_document_template_query`) whose *Name* is `OpenCases` and runs its *Fetch XML*.

Column	Logical name	Meaning
Name	`bvr_name`	The name templates address the query by.
Fetch XML	`bvr_mlot_fetch_xml`	The FetchXML, with `{{…}}` parameters.
Template	`bvr_fk_document_template`	*Optional.* Empty = shared by every template.
Description	`bvr_slot_description`	*Optional.* Note for whoever maintains the query.
Status	`statecode`	Only active queries are resolved.

Example record:

Column	Value
Name	`OpenCases`
Fetch XML	`>>>>filter>>entity>fetch>`
Template	*(empty — shared)*

Used from any template as:

```
{{#each OpenCases}} … {{/each}}
{{#each query:OpenCases}} … {{/each}}
```

Why a record rather than a JSON field

	Query records	JSON on the template
Reuse across templates	Yes — one record, many templates	Copy-paste per template
Editing	A form, with validation and multi-line text	Hand-edited JSON, escaping and all
Views and search	Ordinary Dataverse views	None
Auditing / who changed what	Per record	Whole blob
Security	Per record, standard privileges	Whatever the template record allows
Solution transport	A normal record	Embedded in template data
A broken query	Fails that one query	Invalid JSON breaks **every** query on the template

Shared and template-specific queries

The *Template* lookup is optional and changes nothing for the template author:

- **Empty lookup** → the query is shared. Any template may use it.
- **Set to a template** → the query belongs to that template only.
- If both exist under the same name, the **template-specific one wins**, so a single template can override a shared query without a rename and without affecting anything else.

Resolution and caching

Queries are resolved **lazily by name** and memoised for the run: a template usually references a handful, so loading the whole library up front would read far more than needed. Each distinct name costs one query per generation, however many regions use it.

The query **record** is read with system privileges — it is configuration, like the template itself — so a user needs no access to the query table. The **rows** the query returns follow the run's scope.

7 Availability rules

7.1 Model

Availability is decided in **two steps**, and they live on two different records:

1. **Which table** — the template itself declares it, as *Associated Entity Logical Name*. A template is never offered anywhere else.
2. **Which records of that table** — each Document Generation Availability Rule (`bvr_document_template_availability_rule`) adds **one comparison**: attribute, operator, value.

So a rule carries no entity name of its own. It inherits the table from the template it belongs to, and a template with **no rules** is available for **every** record of its table — rules narrow a template's reach, they are not what grants it.

Column	Logical name	Meaning
Template	`bvr_fk_document_template`	The template the rule belongs to — and, through it, the table the rule applies to.
Attribute Path	`bvr_slot_attribute_path`	Attribute to test. May be a dotted path.
Operator	`bvr_os_operator`	Comparison to apply. See the table below.
Value	`bvr_slot_value`	Value to compare against, written as text.
Group By	`bvr_bool_group_by_and`	**Yes** = AND group; **No** = OR group. See §7.5.
Name, Description	`bvr_name`, `bvr_slot_description`	For people reading the rule; the engine ignores them.
Status	`statecode`	Only active rules are considered.

There is **no query language here**. A rule is evaluated against the record the framework already has in memory, so it needs no round trip of its own and works identically against any table.

7.2 Operators

Operator is a choice with these values:

Option (label)	Value	Meaning
Equal	100310000	Attribute equals the value.
Not Equal	100310001	Attribute differs from the value.
GreaterThan	100310002	Attribute is greater than the value.
LessThan	100310003	Attribute is less than the value.
GreaterEqual	100310004	Attribute is greater than or equal to the value.
LessEqual	100310005	Attribute is less than or equal to the value.
Contains Data	100310006	Attribute has a value. Ignores *Value*.
Does Not Contain Data	100310007	Attribute is empty. Ignores *Value*.

The values are mirrored exactly in `Models/AvailabilityOperator.cs` — the enum member *is* the option value. An option the framework does not know falls back to **Equal** rather than comparing with something arbitrary.

7.3 How a value is compared

Value is text, and it is coerced to the attribute's own type before comparing, so an author writes whatever is natural:

Attribute type	Write	Meaning
Choice	`1`	the option's numeric value
Choice	`Active`	the option's label, in the language of whoever generates the document
Two options	`true` / `false` / `1` / `0`	the flag
Lookup	a GUID	the referenced record
Lookup	any other text	the referenced record's name
Date/time	`2026-01-31`	that date and time for whoever generates the document, compared with the column as it prints (§4.4). `2026-01-31T10:00:00Z` names an exact moment.
Number / money	`1000.50`	invariant decimal
Text	anything	case-insensitive comparison

An empty attribute equals nothing and differs from everything, exactly as in the template language: `Equal` is not satisfied, `NotEqual` **is**, and the ordering operators are not. `ContainsData` / `DoesNotContainData` remain the direct way to say "has a value" / "is empty".

7.4 Dotted attribute paths

Attribute Path may reach through lookups, using the same syntax and the same resolver as the template language:

```
statuscode
ownerid.businessunitid.name
customerid.primarycontactid.emailaddress1
```

The evaluator declares which columns its paths read, on the record and on every table they reach, and records are retrieved with exactly those.

A rule with a mistake in its path — a misspelt column, `name..x`, or a path through a text column — cannot be decided. It does not empty the list for the table: its template is still listed, with a warning icon in the ribbon menu and a warning and the message in *View All Available Templates*, where Preview and Download are disabled for it. Every other template is unaffected. Choosing it from the menu shows the message in a ⚠ *Warning* alert, which asks the user to contact their administrator, and generates nothing. The message names the rule and the template:

```
Availability rule 'nmae' of document template 'Quote': The entity 'account' has no attribute 'nmae' (referenced by 'nmae').
```

Fix the rule's *Attribute Path* and the mark disappears.

7.5 Semantics

Each rule says how it combines with its siblings through **Group By** (`bvr_bool_group_by_and`):

Group By	The rule is in	Combines with
Yes	the **AND** group	the other AND rules, with **AND**
No	the **OR** group	the other OR rules, with **OR**

The template is offered when **either group** is satisfied:

```
(and₁ AND and₂ AND …)  OR  (or₁ OR or₂ OR …)
```

For three rules on the same entity:

Rules	Meaning
all Yes	`a AND b AND c` — every condition must hold
all No	`a OR b OR c` — any condition is enough
`a` Yes, `b` `c` No	`a OR b OR c` — the AND group is just `a`
`a` No, `b` `c` Yes	`a OR (b AND c)`
`a` `b` Yes, `c` No	`(a AND b) OR c`

Also:

- A template with **no rule at all** is available for every record of its table.
- A template is **never** offered on a table other than its own *Associated Entity Logical Name*, whatever its rules say.
- A rule set with **no AND rule** is pure OR: any one rule is enough.
- A rule set with **only** AND rules is pure AND.

7.6 Examples

The table comes from the template's *Associated Entity Logical Name* (shown here for context); everything else is on the rule:

Intent	*template's table*	Attribute Path	Operator	Value	Group By
Any case	`incident`	*(no rule at all)*	—	—	—
Only active cases	`incident`	`statecode`	Equal	`0`	No
Only high priority	`incident`	`prioritycode`	Equal	`High`	No
Only assigned cases	`incident`	`ownerid`	Contains Data	—	No
Only unassigned cases	`incident`	`ownerid`	Does Not Contain Data	—	No
Large accounts only	`account`	`revenue`	GreaterThan	`1000000`	No
One business unit	`incident`	`ownerid.businessunitid.name`	Equal	`Head Office`	No

Intent	*template's table*	Attribute Path	Operator	Value	Group By
Everything except blocked	`account`	`statuscode`	Not Equal	`Blocked`	No
Active and high priority — two rules	`incident`	`statecode`	Equal	`0`	Yes
↳	`incident`	`prioritycode`	Equal	`High`	Yes

7.7 Limitations

A rule set is two groups, not an arbitrary boolean tree: the AND group is AND-ed, the OR group is OR-ed, and the two groups are OR-ed. Nested shapes such as `(a OR b) AND (c OR d)` are therefore not expressible directly — add a calculated or rollup column on the entity and write one rule against it, or express the condition inside the document with `{{#if}}`. Both keep availability a flat, readable list, which is what makes it safe for an administrator to edit.

7.8 Enforcement

Rules are enforced **both** in the ribbon (which templates appear) **and** inside `bvr_GenerateDocumentFromTemplate`, per target record. Hiding a button is not the only thing preventing an ineligible template from being used — the action is callable directly.

Rules decide **which template applies to which record**; they are not a security boundary on their own. See TECHNICAL.md §16 before relying on them to keep data from a user.

8 Setting up templates in Dynamics 365

Everything above is written in Word. This section is the app side: the record that carries the template, when it is in circulation, and where it shows up for the user.

8.1 The template record — Document Generation Template

Column	Logical name	Required	What to put in it
Name	`bvr_name`	✔	The template's name. It is what the ribbon button says, and the generated file is named after it.
Type	`bvr_os_document_type`	✔	**Word** — the only type supported today. A template of any other type is neither listed nor generated.
Associated Entity Logical Name	`bvr_slot_associated_entity_logical_name`	✔	The table the template serves. Choose it from the list, which you can search by display name or logical name. The field shows `Account (account)` and stores only the **logical** name, such as `account`, `incident` or `new_myentity`. This is the only thing that binds a template to a table, and a template is never offered anywhere else. A warning under the field means this environment has no such table, which can happen with a template moved from another environment.
Language	`bvr_int_language_code`	○	The language the template is written in. See §4.4.
Use Current User Selected Language	`bvr_bool_use_current_user_selected_language`	○	**Yes** = render in the language of whoever generates the document, instead of *Language*.
Scope	`bvr_os_scope`	○	Who the document's **data** is read as. **System** (the default, also when empty): the document is complete whoever prints it. **User**: it shows only what the person generating it can see; a row they cannot see is empty, and no access to a whole table stops the document with a message naming it. Either way, the user must be able to open the record itself. See §8.6.
Description	`bvr_slot_description`	○	Shown as the ribbon button's tooltip and in the *View All Available Templates* dialog.
Quick Template	`bvr_bool_quick_template`	○	**Yes** (the default for a new template) = listed directly in the ribbon menu. See §8.4.
Status	`statecode`	✔	Must be **Active**. See §8.3.

8.2 The Word file is an attachment

The `.docx` is **always the newest note attached to the template record** — there is no column holding the document. Two consequences worth knowing:

- **Correcting a template is an upload, not a delete.** Attach the fixed `.docx` as a new note and it takes effect immediately; the previous file stays on the record as history.
- **A template with no attachment cannot render.** The note must carry a file, not just text.

8.3 When a template is usable

A template is in circulation when **both** hold:

Condition	Column
The record is **Active**	`statecode` — *Deactivate* takes a template out of circulation, *Activate* puts it back
It is a **Word** template	`bvr_os_document_type`

An unusable template is **neither listed nor generatable**. The same two conditions are applied when the list of templates is built and again when the document is produced, so a template is never offered and then refused.

To take a template out of service while revising it, deactivate it: it keeps its notes and history and is one click from returning.

8.4 Where a template appears

There are two surfaces, and a template can be eligible for a record without being listed in the first one:

Surface	Shows
The ribbon button's menu	Only templates whose *Quick Template* is **Yes**
View All Available Templates	**Every** template eligible for the record

Quick Template is presentation only — it never affects eligibility, and every template is always reachable through *View All Available Templates*. An empty value counts as No. When no template is a Quick Template, a menu-style button opens *View All Available Templates* directly.

Eligibility itself is decided by the template's table plus the availability rules, and it is enforced when the document is generated as well as when the list is built.

From a grid, a user may select several records and get one document: each record is rendered from the same template and the results are joined by a page break (the default) or a line break. Availability is evaluated against the **first selected record** — a menu can only show one set of buttons — and a selection is capped at 250 records, the largest page a Dynamics grid offers. The server enforces the same cap, whoever calls it.

Headers, footers and page setup in a multi-record document.

- **Page break (default).** Each record starts on a new page and keeps everything its own copy of the template has: every section with its page size, orientation and margins, and its own headers and footers. A `{{bvr_name}}` in the header shows each record's own name on that record's pages, and a *Different First Page* header appears on the first page of **every** record, not only the first page of the file.
- **Page numbers run on** through the whole file (record 2 starts at page 4 if record 1 had 3 pages), even if the template's numbering is set to *Start at 1*. List numbering also continues from one record to the next.
- **Line break.** Records follow each other on the same pages, and a page can only have one header, so headers and footers — and the page setup of the last section — are those of the **first** record. Keep placeholders out of headers and footers in templates meant for line-break output.

8.5 Adding a template, step by step

1. **Create the Document Generation Template record.** Set *Name*, *Type* = Word, *Associated Entity Logical Name* = the table it serves, chosen from the list, *Language* (or *Use Current User Selected Language* = Yes), and *Scope* (§8.6).

2. **Write the `.docx` in Word** and attach it to the record as a note. §3 to §5 are the syntax; the cheat sheet is the one-page version.
3. **Decide whether it belongs in the ribbon menu.** *Quick Template* = Yes lists it there; No keeps it in *View All Available Templates* only.
4. **Add availability rules only if you need to narrow it.** With no rules at all the template is available for **every** record of its table. See §7.
5. **Create any named queries** the template uses — one Active Document Generation Query per `{{#each query:Name}}`. See §6.4.
6. **Test** on one record, then on a multi-record grid selection. There is nothing to configure for the button: installing the managed solution adds **Generate Document** to the forms and home grids of every table (TECHNICAL.md §5.5).

8.6 Scope — whose access a document reflects

Templates and their rules and queries are configuration: they are always read with system access, so users need no privileges on the document generation tables. The **Scope** column decides how the document's **data** is read:

Scope	The data is read as	Choose it when
System *(default)*	The system	A printed form must be complete regardless of who prints it — an invoice, a contract.
User	The person generating it	The document must not show anything the person could not look up themselves.

Under either scope the person must be able to open the record the document is for. A developer calling the action from their own code can read a System template as the user (`PreferredScope = User`), but nothing can make a User template read as system. The ribbon and the dialog always use the template's own Scope.

8.7 The subscription

Document Generation needs an active subscription from the Bever Marketplace, activated once in each environment. Until it is, no template is listed and no document is generated.

Activating. A System Administrator, or a user with the *Document Generation Administrator* role, opens the solution's configuration page (*Document Generation Configuration*), enters the license key and selects **Activate**. The page then shows the state of the subscription, how many environments the license is activated in, and links to **Change Payment Method** and **Cancel Subscription**. **Deactivate** releases this environment from the license, for example before moving it to another one. No one else can activate or deactivate, even by calling the actions directly: Dataverse checks the caller's role. The key is kept in the environment's single *Document Generation Template Configuration* record; a second one cannot be created.

What users see. The subscription is checked every time templates are listed or a document is generated:

Subscription	Documents	*View All Available Templates*
Active, Trial	Generated	As usual
Past Due	Generated	As usual. The configuration page shows *Past Due* and how to fix it
Canceled, Paused, or no valid license	Not generated	A message saying why, with the search and buttons disabled

The ribbon menu shows the same message in an error dialog. Each check asks the Bever Marketplace, so if it cannot be reached, documents cannot be generated until it can.

9 Worked end-to-end example

Goal. A "Client Summary" for an `account`, listing its open cases, flagging large accounts, with a letterhead footer.

9.1 Configuration

Document Generation Template record:

Column	Value
Name	Client Summary
Type	Word
Associated Entity Logical Name	account
Language	English (1033)
Use Current User Selected Language	No
Scope	System
Description	Account overview with open cases
Quick Template	Yes
Status	Active
Note attachment	`ClientSummary.docx`

Document Generation Query record:

Column	Value
Name	OpenCases
Fetch XML	`>>>>>filter>>entity>fetch>`
Template	*(empty — shared with any template that wants it)*
Status	Active

Availability rules: **none** — *Associated Entity Logical Name* = `account` already makes it available to every account. (To restrict it to large accounts, add one rule: *Attribute Path* = `revenue`, *Operator* = GreaterThan, *Value* = `1000000`.)

9.2 The `.docx`

```
                                        CLIENT SUMMARY

Account:        {{name}}
Owner:          {{ownerid.fullname ?? "Unassigned"}}
Business unit:  {{ownerid.businessunitid.name}}
Region:         {{ownerid.businessunitid.parentbusinessunitid.name ?? "—"}}
Revenue:        {{revenue:currency}}
Employees:      {{numberofemployees:N0}}
Segment:        {{If(revenue > 10000000, "Enterprise", revenue > 1000000, "Mid-market", "SMB")}}
Created:        {{createdon:dd/MM/yyyy}}
Contact:        {{primarycontactid.fullname}} ({{primarycontactid.emailaddress1 ?? "no e-mail"}})

{{#if numberofemployees > 500}}
⚠  Large account — a relationship manager review is required before issuing this summary.
{{#else}}
Standard servicing applies.
{{/if}}

OPEN CASES ({{$count}} shown)

┌──────────────────────────────┬──────────────┬──────────────┬──────────────────────┐
│ {{#each OpenCases}}{{$index}}. {{ticketnumber}} │ {{createdon:date}} │ {{prioritycode}} │ {{$parent.name}}{{/each}} │
└──────────────────────────────┴──────────────┴──────────────┴──────────────────────┘

Footer:  Generated by {{$user.fullname}} on {{$now:"dd.MM.yyyy HH:mm"}}
```

9.3 What happens

1. The ribbon calls `bvr_RetrieveAvailableDocumentTemplates` → no rule narrows it, and it is a Quick Template → "Client Summary" appears in the menu.
2. The user clicks it → `bvr_GenerateDocumentFromTemplate` runs, in English, the template's language.
3. The template's markers are read **once**; the needed columns are computed as `name, ownerid, revenue, numberofemployees, createdon, primarycontactid, transactioncurrencyid` — note that `ticketnumber` and friends are *not* requested from `account`, because the collector knows they sit inside a repeating region. The same pass plans the other tables: `fullname, businessunitid` of the owner, `name, parentbusinessunitid` of the business units, `fullname, emailaddress1` of the contact.
4. The account is retrieved with exactly those columns. Its Scope is System, so the data is read as system once the server has checked the user can open the account.
5. `ownerid` → `systemuser` → `businessunitid` → parent unit: **three retrieves**, cached, each with only its planned columns, however many placeholders use them.
6. `OpenCases` runs with `{{$recordid}}` substituted; the table row is cloned per case; `{{$parent.name}}` reaches back to the account.
7. The `{{#if}}` block keeps one branch.
8. The footer renders in the same pass as the body.
9. The file is returned base64 and downloaded as `Client Summary 2026-08-03 14-30-00.docx`.

10 Troubleshooting

A structural problem — a placeholder that does not parse, an attribute that does not exist, a region that is never closed, an unavailable template — stops the document and produces a message naming the placeholder, the template and the record at fault. Start from that message; it usually names the fix.

10.1 What is deliberately *not* an error

A **null value renders as empty**. That is data, not a defect, and a document full of blanks where a record genuinely has no phone number is correct output. Only *structural* problems — things no data could fix — raise. This split is what makes `FailOnUnknownAttribute` meaningful: a typo is structural and should stop the run, an empty field is not.

10.2 Common symptoms

Symptom	Cause	Fix
"There is no valid license in this environment"	No license key is activated, or the marketplace does not recognise it	An administrator activates one on the configuration page. See §8.7.
"The subscription has been canceled" or *"... is paused"*	The subscription is no longer active	Renew or resume it in the Bever Marketplace.
Past Due on the configuration page	The last payment failed	Documents still generate. Use *Change Payment Method* on the configuration page before the subscription is canceled.
"You do not have permission to activate or deactivate the subscription"	The user lacks the role	Ask an administrator.
"This environment already has a configuration record"	Someone tried to create a second one	Change the existing record instead.
"The subscription could not be verified"	The Bever Marketplace did not respond	Try again later.
Ribbon menu says *"No templates available"*	No eligible template is a **Quick Template**	Expected. Set *Quick Template* = Yes, or use *View All Available Templates*, which lists every eligible one.
A template appears in *View All Available Templates* but never in the menu	*Quick Template* is empty or No	Only Yes means listed.
A template shows a warning icon in the menu or the dialog	One of its availability rules has a mistake in its path	The message names the rule. Fix its *Attribute Path*. See §7.4.
A template appears **nowhere**	It is Inactive, its *Type* is not Word, its *Associated Entity Logical Name* is not this table, or its rules do not hold for this record	Check in that order. For rules: every **AND** rule must hold, or at least one **OR** rule. See §8.3 and §7.5.
"Document template ... is not active" / *"... is not a Word template"*	The template was deactivated or its type changed after the menu was built	Reactivate it, or set *Type* to Word.
Every template appears for every record	No active rules exist for them	No rules means "the whole table" — that is the documented default. Check that the rule records exist and are Active.

Symptom	Cause	Fix
"Template … is not available for records of type …"	A rule exists but its comparison is not satisfied	Check *Attribute Path*, *Operator*, *Value* and *Group By* on the rules.
Choice labels are in the wrong language	The document follows *Language* on the template, or the user when *Use Current User Selected Language* is Yes	See §4.4. A language that is not enabled in the environment has no labels and shows the default ones.
`{{x ?? "default"}}` **prints nothing for a blank field**	`??` falls through on **null only**, and the field holds `""`	Use `{{If(x = null, "default", x)}}`. See §3.5.
"has no attribute …"	Typo in the placeholder, or the field belongs to a related entity	Fix the name, or add the lookup hop.
"… is not a lookup and cannot be traversed"	A text column written as if it pointed somewhere, `{{name.x}}`	Remove the part after it, or use the lookup column.
"'query:Q' does not select the column …" **inside a region**	The query's FetchXML does not select that column	Add `>` to the query. See §5.2.
"… reads through the lookup … on a row of 'query:Q'"	`{{ownerid.fullname}}` on a query's row	Add a `link-entity` with an alias to the query and write `{{Alias.fullname}}`. See §5.2.
"The query's link-entity 'User' does not select …"	The column is not among the link's attributes, it has an alias of its own, or the case differs	Write what the query declares, or use `?.` if the column is optional: see §5.2.
"the link-entity '…' selects columns but has no alias"	A `link-entity` without an `alias`	Give it one.
"returned more rows than Dataverse returns at once"	A `distinct` or aggregate query has more than 5000 rows	Narrow its filter or add `top`. See §6.3.
"Invalid template placeholder '…': its format is not valid for this value"	A format the value's type does not accept, such as `{{$now:Q}}`	Fix the format. See §4.
"You do not have permission to read '…' records"	The template's Scope is User, and the person has no access to a table it reads	Grant read access, or change the template's Scope. See §8.6.
"Select at most 250 records"	More than 250 records selected	Select fewer.
An error ending *(record '…' of type '…')*	One record of a multi-record selection failed	The message names which; generate that record alone to see it.
An empty line where a region marker was	Its line also holds a section break, page break or bookmark, which is kept	Expected: see §5.1.
A region returns more rows than `count`	The query's `count` or `page` is empty, for instance from a parameter with no value	An empty `count` is no limit. See §6.3.
A Time Zone Independent or User Local + Date Only value looks a day or hours off	The document follows the column's **Behavior**, not its name	See the table in §4.4: it matches what the same user sees on the form.
Placeholder rendered literally in the output	Unbalanced region, or a `{{` with no `}}`	The error message names the marker.
"the '{{#else}}' marker must start its own paragraph…"	`{{#if}}` / `{{#else}}` in one paragraph	Use inline `If(…)`, or put branches on separate paragraphs.
"returned more than N rows"	Region exceeded `MaxRegionRows`	Narrow the query or raise the limit.

Symptom	Cause	Fix
Template body missing	No note with `isdocument = true` and a non-null `documentbody`	Re-attach the `.docx`.

Symptoms that point at the code, the deployment or the security scope — a document that is thinner for one user than another, formatting lost after substitution, dates an hour off — are in TECHNICAL.md §14.

Appendix — Syntax cheat sheet

```
FIELDS
  {{name}}                                    field
  {{ownerid.businessunitid.name}}             nested lookup, unlimited depth
  {{[new_note^new_caseid].subject}}           oldest related record (1:N)
  {{[new_tag^^new_case_tag].name}}            oldest related record (N:N)
  {{First(MyQuery).new_total}}                first row of a named query
  {{First(query:MyQuery).new_total}}          the same, written explicitly
  {{First(relationship:new_case_notes).subject}}   first related record
  {{ownerid?.internalemailaddress ?? ownerid?.emailaddress}}   ?. — empty, not an error, if the table lacks the column
  {{$parent.name}}                            enclosing record (inside a region)
  {{$parent.$parent.name}}                    two regions out
  {{$root.name}}                              the record the document is generated for
  {{$this.name}}                              explicit current record
  {{CaseStatus}} {{Email}}                    named query: attribute alias
  {{User.fullname}} {{BusinessUnit.name}}     named query: link-entity alias + column
  {{ownerid}} {{ownerid:id}}                  named query: a lookup on the row (nothing beyond it)

VARIABLES
  {{$user.fullname}} {{$now}} {{$today}} {{$recordid}} {{$entityname}} {{$index}} {{$count}}

OPERATORS
  ??   default — null only, "" is a value        +    concatenate
  =  <>  !=  <  <=  >  >=          AND  OR  NOT  ( )
  =  and  <>  read "" as empty;  ??  does not

CONDITIONALS
  {{If(cond, value, default)}}
  {{If(c1, v1, c2, v2, …, default)}}

FORMATS
  {{d:dd/MM/yyyy}} {{d:HH:mm}} {{d:date}} {{d:time}} {{d:"MMMM dd, yyyy"}}
  {{b}} {{b:label}} {{b:text}} {{b:checkbox}} {{b:crosscheckbox}} {{b:checkmark}} {{b:Agreed/Declined}}
  {{n:N2}} {{n:percent}} {{m:currency}} {{m:code}}
  {{c:value}} {{c:label}} {{ms:" / "}}
  {{l:name}} {{l:id}} {{l:entitytype}}
  {{s:upper}} {{s:lower}} {{s:trim}}

LANGUAGE
  {{c:label:lcid=1049}}   {{c:lcid=1033}}   {{d:"d MMMM yyyy":lcid=1033}}   {{n:N2:lcid=1049}}

REGIONS
  {{#each relationship:new_account_cases}} … {{/each}}   preferred
  {{#each MyQuery}} … {{/each}}
  {{#each query:MyQuery}} … {{/each}}
  {{#if cond}} … {{#else}} … {{/if}}

QUERY LIMITS
  <fetch top="2">              first 2 rows
  <fetch count="2">            first 2 rows
  <fetch count="2" page="2">   2 rows of page 2
  <fetch>                      every row
  <fetch distinct="true">      up to 5000 rows, more is an error

QUERY ROWS
  a row holds only what the FetchXML selects; a linked table through a link-entity with an alias

ESCAPING
  \{{        prints {{        \{{#each Items}}   prints {{#each Items}}
```