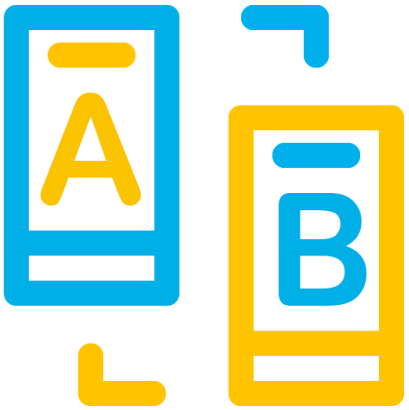
Check Text Changes



Description

Ever pushed out an updated data file and wondered whether it really carried any meaningful change before triggering your whole downstream pipeline?

Check Text Changes is your answer.

It sits in your Switch flow and does one thing brilliantly:

- it compares the text content of two files, a **reference** file and an **asset** and routes the incoming job based on whether differences were detected.
- it understands XML, JSON, and any other text-based format (CSV, TXT, YAML, …), producing a structured diff that downstream tools can act on.

Think of it as a very patient, very precise auditor. It opens both files, inspects them down to the value, and tells you exactly what changed, where, and how ; ready for a mailer, a notification, a report, or a conditional downstream process.

In a nutshell

Check Text Changes compares a reference file against an asset (the job file, an external file, or a dataset attached to the job), computes a detailed diff when they differ, filters the diff by path if requested, and dispatches the job to the Success, Warning, or Error traffic-light output. A Log connection can also receive the detailed diff as an XML or JSON file.

Compatibility

Switch 2025.11 or higher.

Connections

- At least one incoming connection.
- TrafficLight outgoing connection.

Use case

Perfect for

- ERP/MIS workflows where an updated data file must be compared against a reference before reprocessing.
- Order-management flows that only re-trigger downstream steps when the order data has actually changed.
- Print-production flows that detect whether a re-submitted job file differs from the previous version.
- Quality-control processes that need to audit what fields of a document were modified and by how much.
- Notification pipelines that send an email listing the changes detected between two versions of a file.

Common Scenarios

Order change detection in an ERP workflow
The incoming job is a fresh XML export of an order. The reference is the previous version of that order. **Check Text Changes** detects, for example, that /order/items/item[3]/quantity changed from 50 to 120, emits a dataset describing the change, and routes the job to Success so the imposition tool picks it up again with the new data.

Typical Workflow

1. A job arrives in the script element carrying (or referring to) the asset to compare.
2. The script reads the **reference** file and resolves the **asset** according to the selected mode (job file, external file, or job dataset).
3. Both sources are validated: not a directory, not hidden, not a graphic file, matching extensions.
4. The text content is compared and, if differences are found, a detailed diff is computed: XPath-style paths for XML, dot-notation for JSON, line numbers for plain text.
5. If **filtering** is enabled, only changes whose path matches at least one filter pattern are kept.
6. A dataset (XML or JSON) summarising the result and all individual changes is attached to the job.
7. The job is routed to the Success, Warning, or Error output; the detailed diff is also emitted on the matching Log connection.

Flow element properties

- **Reference file to compare**
  Absolute path to the reference file to compare against.
  The file must be a text file (not a directory, not hidden, not a graphic format).
  - **Asset to compare**
    Determines how the asset to be compared is resolved. Three modes:
    - Job is asset
      The incoming job file itself is used as the asset.
    - Job refers to asset
      An absolute path to an external file is provided.
      - Choose file
        Absolute path to the external asset file.
    - Dataset is asset
      A metadata dataset attached to the incoming job is used as the asset. The dataset model must be XML or JSON and must match the extension of the reference file.
      - Dataset name
        Name of the dataset attached to the incoming job that will be used as the asset. The dataset model must be XML or JSON (Opaque, XMP, JDF are rejected) and must match the extension of the reference file.
  - Filter changes
    Enable or disable change filtering.
    When set to Yes, the Filter path property becomes visible. *Default: No.*
    - Filter path
      One filter pattern per line. Only changes whose path matches at least one pattern are kept in the dataset (multi-line = logical OR).
      Other changes are excluded and counted in filteredFromTotal.
      Supported wildcards and anchors:
      - * matches any sequence of characters,
      - ? matches exactly one character,
      - ^ anchors at the start,
      - $ anchors at the end.

      If neither anchor is present, the pattern matches anywhere in the path.
      Path formats:
      - XML → XPath-style with / (e.g. /root/customer/email) ;
      - JSON → dot-notation (e.g. customer.email) ;
      - plain text → line numbers (e.g. line12).

      A single pattern cannot match both XML and JSON paths ; provide one per format when the flow processes both.
      See documentation for more information about filter patterns.

  - Dataset name
    Name of the metadata dataset attached to the outgoing job. *Default: CheckTextChanges.*
  - Dataset type
    Format of the output dataset. Allowed values: **XML** or **JSON**. *Default: XML.*

Outgoing connection

Check Text Changes has three outgoing connections.

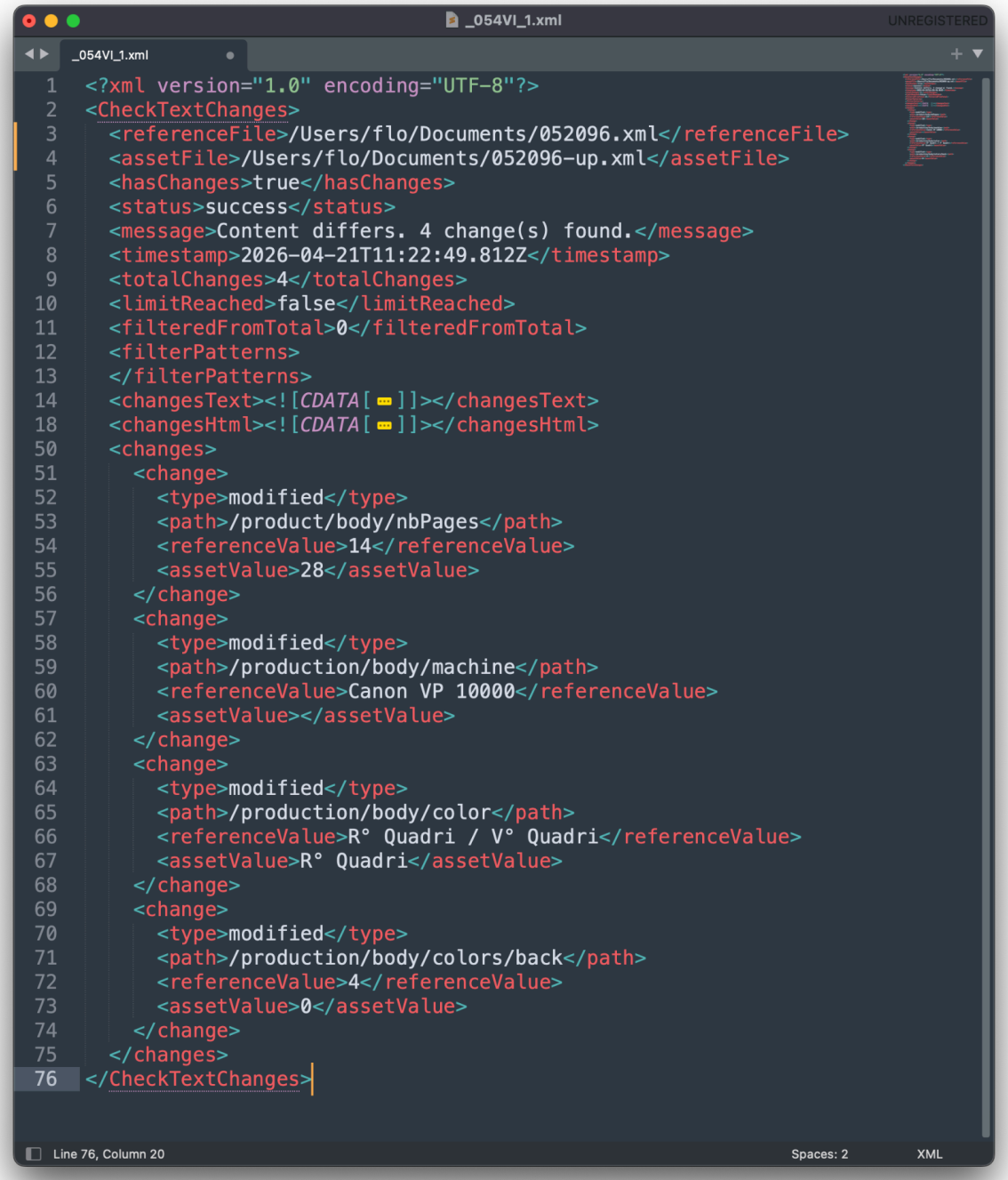
Each carries an XML dataset named `CheckTextChanges` attached to the job(s).

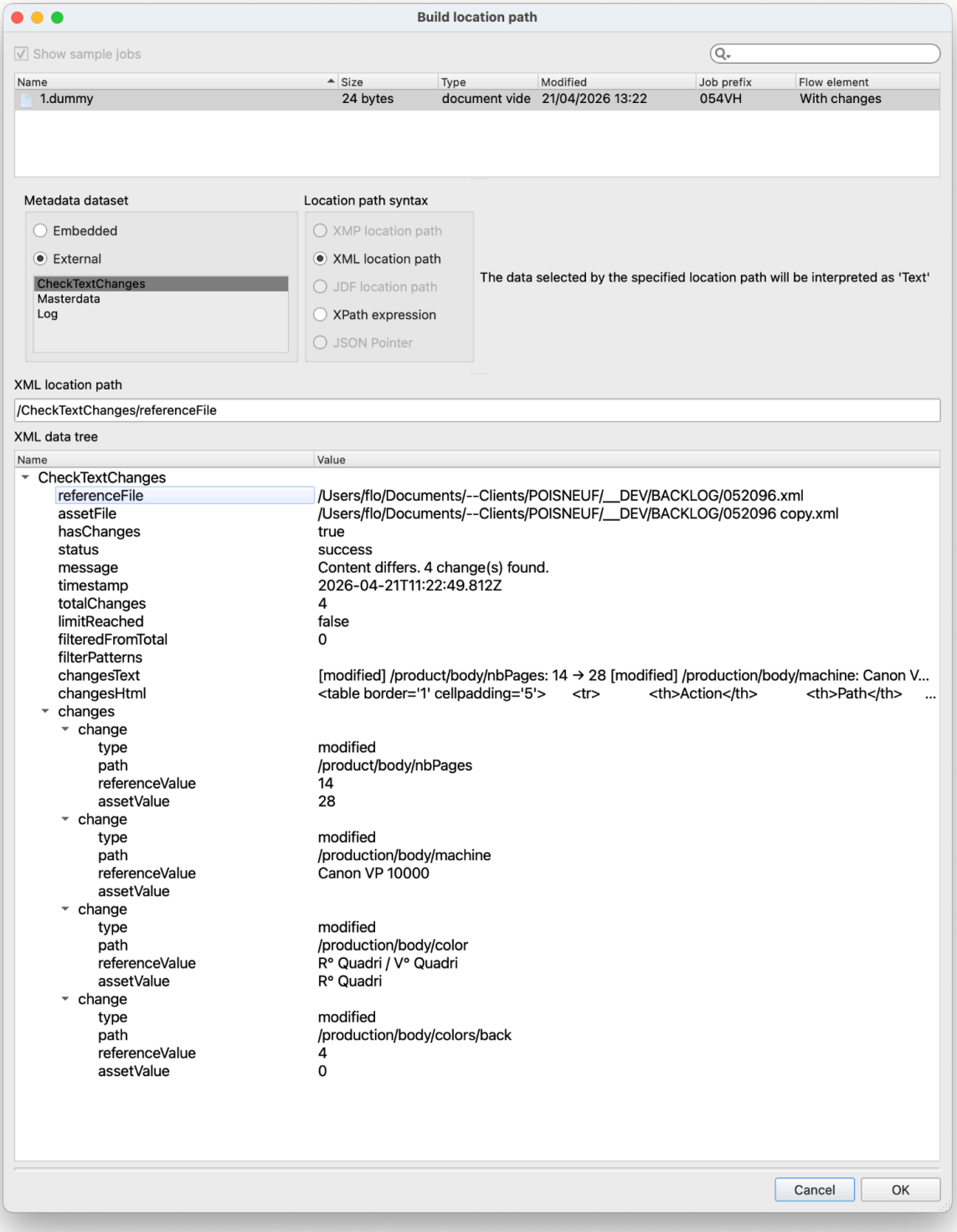
Output behaviour	Output level
Changes detected Triggered when the reference and asset differ and at least one change survives the filter (if filtering is enabled). The `CheckTextChanges` dataset lists every individual change and also contains a pre-formatted text summary (`changesText`) and HTML table (`changesHtml`) ready to embed in emails. The matching Log connection receives the same report as an XML or JSON file named after the incoming job.	**Success**
No changes detected Triggered when the reference and asset are identical, or when filtering is enabled and no change matches the filter patterns. The dataset reports `hasChanges = false` and a human-readable message. Use this connection to skip downstream processing when nothing has really changed.	**Warning**
Comparison failed Triggered on any comparison error: reference or asset file missing, invalid file type (directory, hidden, graphic), extension mismatch, referenced dataset missing or model not matching, read failure, etc. The dataset contains `status = error` and a message explaining the cause. Use this connection for exception handling and operator notifications.	**Error**

Technical Notes

Dataset: CheckTextChanges

Every job leaving **Check Text Changes** through any outgoing connection carries a dataset named `CheckTextChanges`. The same content is emitted as a file on the matching Log connection when wired. Its
format (XML or JSON) follows the **Dataset type** property.




Build location path
Show sample jobs
Name
Size
Type
Modified
Job prefix
Flow element
1.dummy
24 bytes
document vide
21/04/2026 13:22
054VH
With changes
Metadata dataset
Embedded
External
CheckTextChanges
Masterdata
Log
Location path syntax
XMP location path
XML location path
JDF location path
XPath expression
JSON Pointer
The data selected by the specified location path will be interpreted as 'Text'
XML location path
/CheckTextChanges/referenceFile
XML data tree
Name
Value
CheckTextChanges
referenceFile
/Users/flo/Documents/--Clients/POISNEUF/__DEV/BACKLOG/052096.xml
assetFile
/Users/flo/Documents/--Clients/POISNEUF/__DEV/BACKLOG/052096 copy.xml
hasChanges
true
status
success
message
Content differs. 4 change(s) found.
timestamp
2026-04-21T11:22:49.812Z
totalChanges
4
limitReached
false
filteredFromTotal
0
filterPatterns
changesText
[modified] /product/body/nbPages: 14 → 28 [modified] /production/body/machine: Canon V...
changesHtml
<table border='1' cellpadding='5'> <tr> <th>Action</th> <th>Path</th> ...
changes
change
type
modified
path
/product/body/nbPages
referenceValue
14
assetValue
28
change
type
modified
path
/production/body/machine
referenceValue
Canon VP 10000
assetValue
change
type
modified
path
/production/body/color
referenceValue
R° Quadri / V° Quadri
assetValue
R° Quadri
change
type
modified
path
/production/body/colors/back
referenceValue
4
assetValue
0
Cancel
OK

Field	Description
`/CheckTextChanges/referenceFile`	Absolute path to the reference file that was used.
`/CheckTextChanges/assetFile`	Absolute path to the asset (file path or dataset reference).
`/CheckTextChanges/hasChanges`	`true` if differences were found (after filtering), `false` otherwise.
`/CheckTextChanges/status`	`success`, `warning`, or `error` depending on the outgoing connection.
`/CheckTextChanges/message`	Human-readable description of the result.
`/CheckTextChanges/timestamp`	ISO 8601 timestamp of when the comparison was performed.
`/CheckTextChanges/totalChanges`	Number of changes after filtering. Capped at 50 (see Change limit).
`/CheckTextChanges/limitReached`	`true` if the 50-change limit was reached.
`/CheckTextChanges/filterPatterns`	List of filter patterns applied (empty when filtering is disabled).
`/CheckTextChanges/filteredFromTotal`	Number of changes excluded by the filter (0 if filtering is disabled).
`/CheckTextChanges/changesText`	Pre-formatted multi-line text summary of the changes, suitable for plain-text emails.
`/CheckTextChanges/changesHtml`	Pre-formatted HTML table of the changes, suitable for HTML emails.
`/CheckTextChanges/changes/change`	Individual change entry, one element per change. Each has type (modified / added / removed), path, referenceValue, and assetValue.

Diff strategies

Check Text Changes picks the right comparison strategy based on the file extension.

File type	Extensions	Strategy
XML	.xml, .xhtml, .xsl, .xslt, .xsd, .svg, .rss, .atom, .rdf, .jdf, .html, .htm	Structural, recursive node comparison; XPath-style paths.
JSON	.json	Structural, recursive key comparison; dot-notation paths.
Plain text	.txt, .csv, .tsv, .yaml, .yml, .log, .cfg, .ini, .md, …	Line-by-line diff; path format is `line N`.

Change limit

A maximum of **50 changes** are recorded in the dataset. If more are detected, the `limitReached` flag is set to `true` and `totalChanges` reports the real count. This keeps the dataset lightweight even for files that diverge heavily.

CDATA handling in XML

XML text values are preserved **as they appear in the source**, including any `<![CDATA[…]]>` markers. A value encoded as plain text and the same value encoded in CDATA are therefore reported as different. Use identical encoding on both sides if semantic equivalence is required.

The parser correctly handles split CDATA sequences e.g. `<![CDATA[foo]]]]><![CDATA[>bar]]>`) and closing-tag-like sequences contained inside CDATA blocks ; they do not interfere with element boundaries.

Tips for Success

Keep your reference file up to date

The reference is the baseline against which every incoming job is compared. If the reference drifts out of sync with reality, every comparison will flag changes that are not actually changes. In flows where the reference itself evolves, use a Copy/Move element downstream of the Success output to refresh it with the latest accepted asset.

Use the right asset mode

Job is asset is the simplest choice when the incoming file *is* the thing to compare. **Job refers to asset** is for flows where the job is metadata pointing to a file elsewhere. **Dataset is asset** is powerful when the asset lives as a dataset alongside a job file: you can then compare the dataset against a reference without touching the job file at all.

Design filter patterns per format

The filter engine performs literal matching between your patterns and the change paths. XML paths use `/` separators and JSON paths use `.` separators ; they are not interchangeable. When a flow processes both formats, provide both patterns in the multi-line field, one per line. Use `*` for wildcards and `^/$` anchors when you need exact matches.

Wire the Log connection for rich diagnostics

The Log outgoing connection receives the full diff report as a standalone XML or JSON file named after the incoming job. Route it to an archive folder, a Send Email element, or a Hold element for QA review. It is the fastest way to answer "what changed in this version?" without opening the two source files side by side.

Send rich change notifications by email

The dataset includes two pre-formatted summaries of the changes: `changesText` (plain-text multi-line) and changesHtml (HTML table). In a downstream Send Email element, pull the HTML table directly via an XPath expression into the body of your HTML email, and you get a ready-to-read change report with no scripting required.

Monitor the Error connection

A job hitting the Error output usually points to a configuration or data issue: the reference path is wrong, the asset extension does not match, or the referenced dataset is missing. Route this connection to a problem folder or a notification step so operators see these cases early. The dataset message explains the exact cause.

Master your filter patterns

Filtering is the single most powerful lever to turn **Check Text Changes** into a precision tool rather than a blunt change detector.
A well-crafted filter tells you only about the changes you actually care about, and routes the job to `Warning` when nothing meaningful has changed.
This section covers the full pattern syntax, the three path formats you will encounter, and the single most common pitfall when processing mixed XML/JSON flows.

How matching works

When Filter changes is set to `Yes`, the Filter path property accepts one pattern per line.
Only changes whose path matches at least one pattern are kept in the dataset; other changes are excluded.
Multi-line patterns act as a logical `OR`: a change is kept as soon as any one line matches.
The excluded count is reported in the `filteredFromTotal` field of the dataset so you always know how aggressive the filter was.

Wildcards and anchors

Patterns support four meta-characters.
Every other character (including `/`, `.`, `[`, `]`, letters and digits) is a literal.

- `*` matches any sequence of characters, including zero characters.
- `?` matches exactly one character.
- `^` at the beginning of a pattern anchors the match at the start of the path.
- `$` at the end of a pattern anchors the match at the end of the path.

If neither `^` nor `$` is present, the pattern matches anywhere inside the path.
This is the most common form and typically what you want when filtering by subtree.

XML path format: XPath-style with /

For XML files (.xml, .xhtml, .xsl, .xslt, .xsd, .svg, .rss, .atom, .rdf, .jdf, .html, .htm), paths use / as separator and start with a leading slash.
Element attributes are prefixed with @; repeated sibling elements are indexed in square brackets.

Given this XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <customer>
        <email>user@example.com</email>
    </customer>
    <order status="new">
        <items>
            <item>First</item>
            <item>Second</item>
        </items>
    </order>
    <user name="John" />
</root>
```

The diff paths will look like:

- `/root/customer/email`,
- `/root/order/@status`
- `/root/order/items/item[0]`
- `/root/order/items/item[1]`
- `/root/user/@name`

Example XML filter patterns:

- `/root/customer/*` : keeps every change under /root/customer/ (email, address, phone, etc.).
- `^/order/items/item[0]$` : keeps only the exact path /order/items/item[0]. The brackets are literal.
- `/root/customer/*` on one line plus `/root/order/price` on another : keeps all customer-subtree changes or the specific order price change.
- `@status$` : keeps any change whose path ends with a status attribute (e.g. `/order/@status`, `/item/@status`). Useful for auditing state transitions across unrelated parts of the document.

JSON path format: dot-notation

For JSON files (.json), paths use `.` as separator, without a leading dot.
Array elements are indexed in square brackets, same as XML.

Given this JSON:

```
{
    "customer": { "email": "user@example.com" },
    "items": [
        { "sku": "A", "price": 10 },
        { "sku": "B", "price": 20 }
    ],
    "order": { "total": 100 }
}
```

The diff paths will look like:

- `customer.email`
- `items[0].sku`
- `items[0].price`
- `items[1].sku`
- `items[1].price`
- `order.total`

Example JSON filter patterns:

- `customer.*` : keeps every change under customer. (email, name, address, etc.).
- `^items[0].sku$` : keeps only the exact path items[0].sku.
- `customer.email` on one line plus `order.total` on another : keeps the email change or the order total change.
- `items[*].price` : keeps price changes on any array item under items. The * inside brackets still acts as the wildcard; the brackets themselves are literal.

Plain text path format: line numbers

For any other text file (.txt, .csv, .log, .yaml, .ini, etc.), the diff is computed line-by-line and each change path has the form `line N` where N is a 1-based line number.

Example plain text filter patterns:

- `^line 12$` : keeps only the change on line 12.
- `line 1?` : matches line 10, line 11, …, line 19 (one digit after the 1).
- `^line 1*` : matches any line number starting with 1 (line 1, line 10, line 100, line 123, etc.).

Cross-format filtering: the one gotcha

The filter engine performs literal string matching between your pattern and the change path.
The separators `.` and `/` are treated as literal characters; they are not equivalent.
A single pattern therefore cannot match both an XML path and a JSON path at the same time.

If the same **Check Text Changes** element sits in a flow that processes both XML and JSON files, provide one pattern per format in the multi-line field:

```
/customer/email
customer.email
```

The first line handles XML paths; the second handles JSON paths. Because the filter uses OR semantics across lines, any change matching either line is kept, regardless of the file type being processed.

What happens when the filter excludes everything

If filtering is enabled and no change matches any pattern, the job is routed to `Warning` rather than `Success`: the script considers that, from your point of view, nothing meaningful has changed.
The dataset is attached with an empty changes list and a message explaining that all differences were excluded by the filter.
The original number of unfiltered changes is still visible through the `filteredFromTotal` field, which is useful when tuning your patterns in production.

Example flows