Report Writer URL API — Embedding Reports in a Web Page

This guide is for **developers** who want to embed a Sheerpower Report Writer report directly into a web page, or generate one from a URL, without using the interactive Report Writer UI. A saved report definition can be rendered straight from a single URL, dropped into an `<iframe>`, and narrowed with extra filters — all by query parameters.

Table of Contents

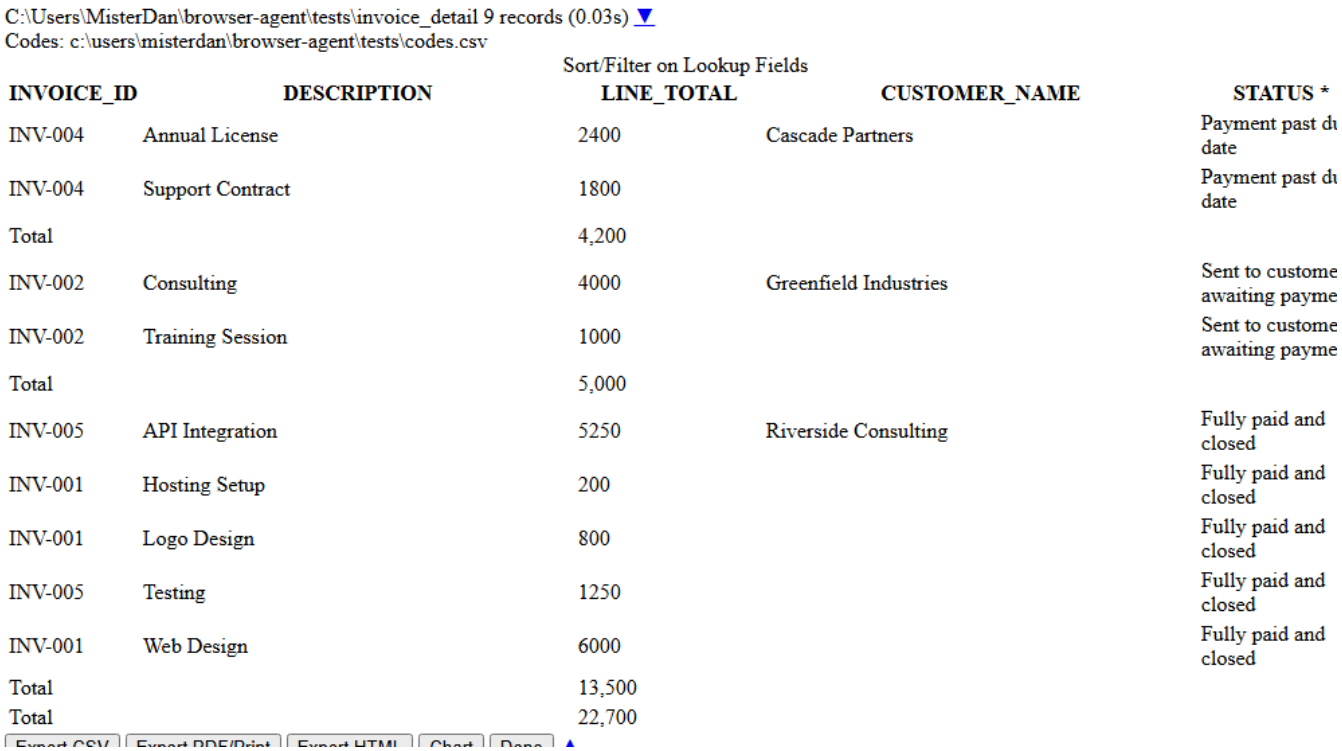
- Quick Start
- The Endpoint
- Parameters
- Saved Definitions (`format`)
- URL Filters (`filter1` … `filter5`)
    - Filter syntax
    - Operators
    - Field names
    - Values and quoting
    - Combining filters (AND)
    - Date ranges
    - Safety
- Auto-Export (`csv`, `pdf`)
- Embedding in an iframe
- Worked Examples
- Building the URL in JavaScript
- Hidden Links (`api`) — *optional*
- Signed Filters (`hash`) — *optional*
- Error Handling
- Styling the Embedded Report
- Live Demo Page
- Reference Card

Quick Start

Embed a saved report named `sales_by_region` in any page:

```
<iframe
  src="/scripts/spiis.dll/report/report_writer.html?action=run&format=sales_by_region"
  style="width:100%; height:600px; border:0">
</iframe>
```

That is the whole integration. The report renders inside the frame:


C:\Users\MisterDan\browser-agent\tests\invoice_detail 9 records (0.03s) ▼
Codes: c:\users\misterdan\browser-agent\tests\codes.csv

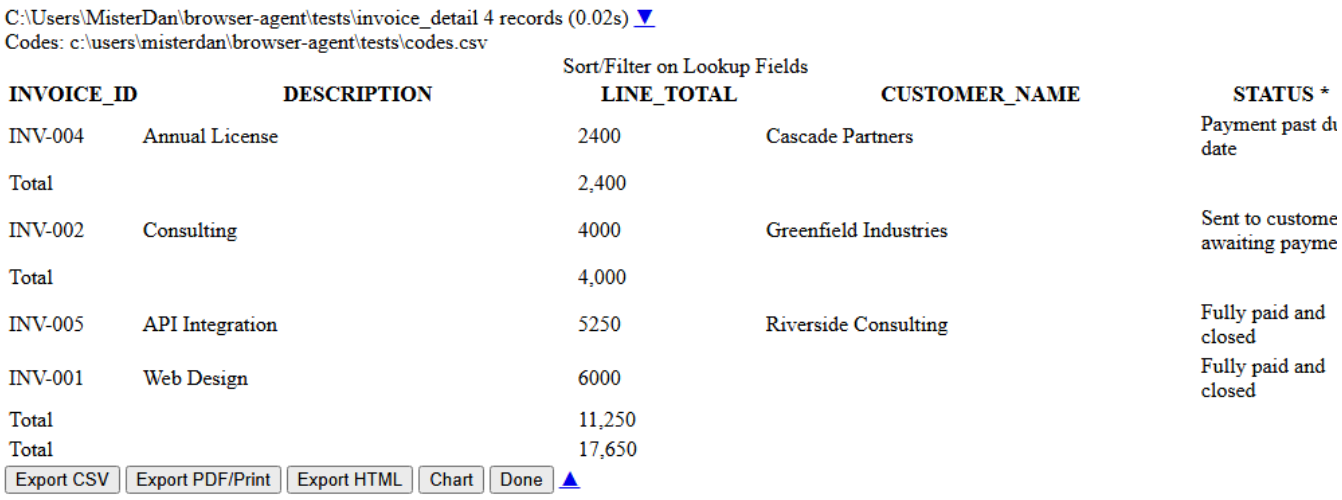
Sort/Filter on Lookup Fields

INVOICE_ID	DESCRIPTION	LINE_TOTAL	CUSTOMER_NAME	STATUS *
INV-004	Annual License	2400	Cascade Partners	Payment past du date
INV-004	Support Contract	1800		Payment past du date
Total		4,200		
INV-002	Consulting	4000	Greenfield Industries	Sent to custome awaiting payme
INV-002	Training Session	1000		Sent to custome awaiting payme
Total		5,000		
INV-005	API Integration	5250	Riverside Consulting	Fully paid and closed
INV-001	Hosting Setup	200		Fully paid and closed
INV-001	Logo Design	800		Fully paid and closed
INV-005	Testing	1250		Fully paid and closed
INV-001	Web Design	6000		Fully paid and closed
Total		13,500		
Total		22,700		

Export CSV | Export PDF/Print | Export HTML | Chart | Done ▲

Add a final filter to narrow it — show only line items of 2000 or more:

```
<iframe
  src="/scripts/spiis.dll/report/report_writer.html?
action=run&format=TEST_report&filter1=LINE_TOTAL>=2000"
  style="width:100%; height:600px; border:0">
</iframe>
```


C:\Users\MisterDan\browser-agent\tests\invoice_detail 4 records (0.02s) ▼
Codes: c:\users\misterdan\browser-agent\tests\codes.csv

Sort/Filter on Lookup Fields

INVOICE_ID	DESCRIPTION	LINE_TOTAL	CUSTOMER_NAME	STATUS *
INV-004	Annual License	2400	Cascade Partners	Payment past du date
Total		2,400		
INV-002	Consulting	4000	Greenfield Industries	Sent to custome awaiting payme
Total		4,000		
INV-005	API Integration	5250	Riverside Consulting	Fully paid and closed
INV-001	Web Design	6000		Fully paid and closed
Total		11,250		
Total		17,650		

Export CSV | Export PDF/Print | Export HTML | Chart | Done ▲

The Endpoint

```
/scripts/spiis.dll/report/report_writer.html
```

- Use a **relative URL** when the host page is served from the same server as the Report Writer (recommended — avoids cross-origin issues). Use the absolute form `http://YOUR-SERVER/scripts/spiis.dll/report/report_writer.html` otherwise.
- Always pass `action=run` to render the report directly (no interactive UI).
- The response is a standards-compliant **HTTP 200** with `Content-Type: text/html; charset=utf-8`, so it is consumable by **any** HTTP client (a browser `<iframe src>`, `fetch()`, `curl`, .NET, etc.) — not just a browser. The body is an **HTML fragment** (the report markup), meant to be loaded into an `<iframe>` or inserted into a page; it is intentionally not a full `<html>` document.

Parameters

Parameter	Required	Value	Purpose
`action`	**yes**	`run`	Render the report (no interactive UI).
`format`	**yes**	saved definition name	Which `.spreport` definition to run (e.g. `gpa`, `sales_by_region`).

Parameter	Required	Value	Purpose
`filter1`…`filter5`	no	`FIELD op VALUE`	Up to five extra **final** filters, ANDed onto the definition's own filters.
`api`	*alt.*	a hidden-link GID	Opaque link: the whole run is loaded from a stored request, so `format` / `filter1`… / `csv` / `pdf` come from the server, not the URL. See Hidden Links.
`hash`	*cond.*	`phash$` signature	**Required only when** an `*api_key*` is configured in `report_writer_security.csv`; signs the filter values. See Signed Filters.
`csv`	no	`auto` or a filename	Skip the HTML and auto-download the report as CSV.
`pdf`	no	any value (e.g. `1`)	Render the report and auto-open the browser print dialog (choose *Save as PDF*).

When `api=GID` is present it supplies the **entire** request — `format` and the other params are taken from the stored link, not the URL (any on the URL are ignored). Otherwise `format` is required.

Only a literal space, `&`, `#`, or `+` inside a filter **value** needs URL-encoding — operators, `|`, and commas do not (see URL Filters).

Saved Definitions (`format`)

The `format` parameter names a report definition that was previously saved in the Report Writer. To create one:

1. Open the Report Writer, pick a table, choose columns, sorts, filters, totals, etc.
2. In **Save Report Definition**, enter a name (e.g. `Sales by Region`) and **Save**.
3. The definition is saved to `@spreports\<name>.spreport` on the server.

Default folder — `@spreports\`. A plain `format=NAME` resolves to `@spreports\NAME.spreport` — the `spreports` folder in the same directory as the `report_writer_web.spsrc` program (the leading `@` means "relative to the program"). Put plain-named definitions there.

Another folder — `format=logical:name`. Give `format` a `logicalname:name` value to read the definition from a folder mapped in `@report_writer_security.csv`, an allow-list beside the program. A logical is accepted **only** if it is listed there; its path then replaces the logical. The CSV has a header row and the columns `logical_name`, `path`, `security`:

```
logical_name,path,security
reports,d:\shared\reports\,read
```

With that entry:

```
?action=run&format=reports:q4_sales   ->  d:\shared\reports\q4_sales.spreport
```

A logical that is not listed — or any logical at all when the CSV is absent — is rejected with `Error: unknown logical '…'`. A `format` value with no `:` always uses the default `@spreports\` folder and never consults the CSV.

The same CSV also holds one optional **special row,** `*api_key*`**,** that turns on signed run URLs — see Signed Filters. (A `format` name can never start with `*`, so it can never collide with a folder logical.)

Reference it by that name. On Windows the name is **case-insensitive** (`format=sales_by_region` and `format=Sales_by_Region` resolve to the same file).

Special name: `last_report` is auto-saved every time a report is generated in the UI, so `format=last_report` always renders the most recent one.

The saved definition carries its own table, columns, sorts, **filters**, totals, lookups, code translations, and formatting — the URL just runs it.

URL Filters (`filter1`…`filter5`)

`filter1` through `filter5` add **extra filters at run time**, on top of whatever filters the saved definition already contains. This lets one saved definition serve many embeds — e.g. the same "Invoices" report scoped per customer, per status, or per date range from the embedding page.

Every operator the Report Writer itself offers is supported.

Two filter syntaxes

1. Inline — for the six comparisons, write `FIELD<op>VALUE` directly:

```
filter1=LINE_TOTAL>=2000
filter1=invoice(STATUS)=Paid
```

2. Pipe form — `FIELD|OP|VALUE[|VALUE2]` — required for the multi-character operators (`contains`, `startswith`, `endswith`, `range`, `anyof`), and usable as an explicit form for the six comparisons too:

```
filter1=DESCRIPTION|contains|design
filter1=LINE_TOTAL|range|1000|4000
filter1=invoice(STATUS)|anyof|Paid,Sent
filter1=LINE_TOTAL|>=|2000
```

> **You do not need to URL-encode the operators.** `=` `<>` `>` `<` `>=` `<=`, the pipe `|`, and commas all pass through a query string unencoded, and the server splits each parameter on its first `=` (so `invoice(STATUS)=Paid` arrives intact). Only a literal space, `&`, `#`, or `+` in a *value* needs encoding (`%20`, `%26`, `%23`, `%2B`). `encodeURIComponent()` is always safe.

Operators

Operator	Meaning
`=`	equals
`<>`	not equal
`>`	greater than
`<`	less than
`>=`	greater or equal
`<=`	less or equal
`contains`	substring (case-insensitive)
`startswith`	begins with
`endswith`	ends with
`range`	inclusive between two bounds
`anyof`	matches any in a comma list

There is exactly one spelling per operator (no aliases). Any other operator is **reported as an error** (`filter1: unknown operator '…'`). Examples (the `|` is a literal character in the URL — no escaping):

```
filter1=invoice(STATUS)=Paid                (inline comparison)
filter1=LINE_TOTAL>=1000
filter1=DESCRIPTION|contains|design
filter1=DESCRIPTION|startswith|Web
filter1=invoice(CUSTOMER_NAME)|endswith|Inc
filter1=LINE_TOTAL|range|1000|4000
filter1=invoice(STATUS)|anyof|Paid,Sent,Draft
```

`range` takes two values (`|LOW|HIGH`); either side may be left blank for an open-ended bound (`range|1000|` = "1000 or more", `range||4000` = "up to 4000"). `anyof` takes a comma-separated list and matches a row if the field equals any item (case-insensitive). Text matching for `contains` / `startswith` / `endswith` / `anyof` ignores case.

Field names

Use the field as it appears in the report's table. Three forms are accepted:

- **Bare field name:** `LINE_TOTAL`, `DESCRIPTION`, `QUANTITY`.
- **Table-qualified:** `invoice_detail(LINE_TOTAL)` — the table name before the parentheses must be the report's own table (it is validated; a wrong name is reported, see below), and the field inside must exist on it.
- **Lookup field:** `invoice(STATUS)` — a field that comes from a table connected to the report's main table. Just name it; the report automatically fetches the matching related record for each row to apply the filter.

A field that does not exist in the report's schema is **reported as an error** (`filter1: unknown field 'X'`) rather than rendering a misleadingly unfiltered report. A table-qualified name whose table is not the report's own table is reported the same way (`filter1: unknown table 'X' (this report's table is 'Y')`).

Values and quoting

- **Numeric fields** — give a bare number: `filter1=QUANTITY>=10`. A non-numeric value on a numeric field is reported (`filter1: 'abc' is not a number for QUANTITY`).
- **Text fields** — give the text unquoted; the API wraps it in quotes for you: `filter1=DESCRIPTION=Widget` becomes the comparison `… = "Widget"`. Embedded double-quotes are stripped, so the value can never break out of the string.
- **Missing value** — an operator with no value (`filter1=LINE_TOTAL>`) is reported (`filter1: missing value`); an empty `range` / `anyof` likewise (`range needs a low and/or high value`, `anyof needs a comma-separated list of values`).

Combining filters (AND)

Multiple `filtern` are **ANDed together** and ANDed with the definition's own filters. A row must pass *all* of them to appear:

```
…&filter1=LINE_TOTAL>=1000&filter2=invoice(STATUS)=Paid
```

→ paid line items of 1000 or more.

Date ranges

A date stored in a sortable text format — like the invoice's `YYYYMMDD` `invoice(INVOICE_DATE)` — compares correctly as text, so a date range is either a single `range` filter or two comparison filters:

```
…&filter1=invoice(INVOICE_DATE)|range|20250101|20251231
…&filter1=invoice(INVOICE_DATE)>=20250101&filter2=invoice(INVOICE_DATE)<=20251231
```

→ rows dated in calendar year 2025 (equivalent).

Safety

URL filters are **safe by construction** — nothing from the URL is executed as code:

- the **field** must match the table schema,
- the **operator** must be one of the whitelisted operators above,
- the **value** is numeric-validated or wrapped as a quoted string (embedded quotes stripped), so text and `anyof` filters only ever compare the field against that literal value — an equality or substring match, never executable code.

So an injection-shaped value is treated as literal data or rejected as an error — never run:

```
filter1=pass("del *")  ->  filter1: could not parse 'pass("del *")' as FIELD op VALUE
```

To hide these filters entirely so a viewer can't see or edit them, use Hidden Links.

Auto-Export (`csv`, `pdf`)

Add to a `…action=run&format=…` URL:

- `csv=auto` — skips the HTML and downloads the report data as a CSV file (data rows only; subtotal/stat rows are excluded). `csv=myreport` names the file `myreport.csv`.
- `pdf=1` — renders the report and automatically opens the browser's print dialog (landscape, report-only stylesheet); choose *Save as PDF* for a PDF.

Filters apply to exports too:

```
/scripts/spiis.dll/report/report_writer.html?
action=run&format=invoices&filter1=STATUS=Paid&csv=paid_invoices
```

Embedding in an iframe

A minimal, responsive embed:

```
<iframe id="rw"
  src="/scripts/spiis.dll/report/report_writer.html?action=run&format=TEST_report"
  style="width:100%; height:70vh; border:0"
  title="Report"></iframe>
```

Change the report at run time by swapping the `src`:

```
document.getElementById('rw').src =
  '/scripts/spiis.dll/report/report_writer.html'
  + '?action=run&format=TEST_report'
  + '&filter1=' + encodeURIComponent('invoice(STATUS)=Paid');
```

A reusable host page that forwards `format` and `filter1..5` from its own query string into the iframe is included as `report_iframe_host.html`:

```
/report_iframe_host.html?format=TEST_report&filter1=LINE_TOTAL>=2000
/report_iframe_host.html?format=TEST_report&filter1=invoice(STATUS)|anyof|Paid,Sent
```

Worked Examples

Assume a saved definition `TEST_report` over an `invoice_detail` table with an `invoice` lookup (fields: `INVOICE_ID`, `DESCRIPTION`, `LINE_TOTAL`, `invoice(CUSTOMER_NAME)`, `invoice(STATUS)`, `invoice(INVOICE_DATE)`).

All operators work unencoded — paste these as-is (each starts with the endpoint `…?action=run&format=TEST_report`):

- **Whole report** — `…?action=run&format=TEST_report`
- **Line items ≥ 2000** — `…&filter1=LINE_TOTAL>=2000`
- **Only paid (lookup field)** — `…&filter1=invoice(STATUS)=Paid`
- **Paid and ≥ 1000** — `…&filter1=invoice(STATUS)=Paid&filter2=LINE_TOTAL>=1000`
- **Amount range 1000–4000** — `…&filter1=LINE_TOTAL|range|1000|4000`
- **Description contains "design"** — `…&filter1=DESCRIPTION|contains|design`
- **Status is Paid or Sent** — `…&filter1=invoice(STATUS)|anyof|Paid,Sent`
- **Customer name starts with "Acme"** — `…&filter1=invoice(CUSTOMER_NAME)|startswith|Acme`
- **Date range (Aug–Dec 2025)** — `…&filter1=invoice(INVOICE_DATE)|range|20250801|20251231`
- **Download paid as CSV** — `…&filter1=invoice(STATUS)=Paid&csv=paid`
- **Print to PDF** — `…&pdf=1`

Building the URL in JavaScript

```
// Build a run URL from a format name and an array of {field, op, value[, value2]}.
// Comparisons use the inline form; multi-character operators use the pipe form.
function reportURL(format, filters) {
  var SYM = { '=':1, '<>':1, '>':1, '<':1, '>=':1, '<=':1 };
  var base = '/scripts/spiis.dll/report/report_writer.html?action=run'
           + '&format=' + encodeURIComponent(format);
  (filters || []).slice(0, 5).forEach(function (f, i) {
    if (!f.field) return;
    var expr = SYM[f.op]
      ? f.field + f.op + f.value                                  // FIELD>=VALUE
```

```
      : f.field + '|' + f.op + '|' + (f.value || '')              // FIELD|op|VALUE
        + (f.value2 != null ? '|' + f.value2 : '');               //   [|VALUE2]
    base += '&filter' + (i + 1) + '=' + encodeURIComponent(expr);
  });
  return base;
}

// Example:
document.getElementById('rw').src = reportURL('TEST_report', [
  { field: 'invoice(STATUS)', op: 'anyof',    value: 'Paid,Sent' },
  { field: 'LINE_TOTAL',      op: 'range',    value: '1000', value2: '4000' },
  { field: 'DESCRIPTION',     op: 'contains', value: 'design' }
]);
```

Hidden Links (`api`) — *optional*

> **Optional, and the strongest lock-down.** Reach for it when the URL should reveal **nothing** — not the report name, not the filters — and when you want links you can **expire or revoke**. For a public or already-scoped report you don't need it. (If you can't mint links on the server, Signed Filters are the lighter alternative — they keep the filters visible but tamper-proof.)

A hidden link is just an opaque id:

```
?action=run&api=GID
```

Everything that would normally follow `action=run` — `format`, the `filter1…filter5`, `csv`, `pdf` — is stored **server-side** under that `GID` (a 30-character, unguessable, URL-safe token) and never appears in the URL. A visitor sees only the id, so they can't read your filters or business logic — and **can't change them**: any extra param tacked onto the URL (a stray `&filter1=…`) is ignored, because the stored request is the sole source of truth.

Creating a hidden link — `make_api_url`

The bundled console tool `make_api_url.spsrc` turns any run URL into a hidden link. Run it, paste the full run URL, and it stores everything after `?action=run`, prints the opaque link, and copies it to your clipboard:

```
Paste the full run URL (must contain ?action=run), then press ENTER:
…/report_writer.html?action=run&format=TEST_report&filter1=LINE_TOTAL%3E%3D2000

API URL:
…/report_writer.html?action=run&api=20260610DgREaGMd2UKhR4Ngdp-7ww

Copied to the clipboard.
```

Hand that `…?action=run&api=GID` URL out — embed it, link it, share it — and the original report and filters stay hidden on the server.

How it resolves

On each request the server looks the `GID` up, checks the link is **not revoked, not expired, and under its use limit**, bumps its usage counters, and replays the stored request internally — rendering exactly as if the original full URL had been requested. An unknown, revoked, expired, or used-up link returns a clean error panel.

Why it's secure

- **Nothing leaks.** The report definition and the filters live server-side; the URL is just an opaque id.
- **Tamper-proof.** Only the stored request runs; extra URL params are ignored, so a visitor can't re-scope the report.
- **Expirable / revocable / one-time.** Each link is a stored record with `expires`, `max_uses` / `use_count`, and `revoked` fields, so a link can be given an expiry date, capped to a single use, or revoked outright. *(`make_api_url` creates a basic never-expiring link; set those fields on the link's record to expire, limit, or revoke it.)*
- Still a **capability URL** — anyone holding the link can view that report until it expires or is revoked. Treat the link as a secret and serve over **HTTPS**.

Requiring hidden links — `*api_required*`

By default the server still answers direct run URLs and the builder UI on every port. To **force all public traffic through hidden links**, add an `*api_required*` row to `report_writer_security.csv`:

```
logical_name,path,security
*api_key*,…,
*api_required*,,
```

The row's mere presence enables it (the other columns are ignored). With it set, the server **rejects any request that doesn't carry an `api=` symbol** — a direct `?action=run&format=…`, the builder UI, a probe — with an "access restricted" panel.

The one exception is **port 4**: SPINS only accepts port-4 traffic from `127.0.0.1`, so a port-4 caller is a trusted local admin and keeps full access — the builder UI, direct runs, everything. On the box itself you build reports and mint links as usual via `http://127.0.0.1:4/…`, while the outside world can only open the opaque links you hand out.

This pairs naturally with hidden links: mint locked, expirable `api=GID` URLs for the reports you want to expose, and `*api_required*` guarantees nothing else is reachable publicly.

Hidden links vs. signed filters

	Hidden link (`api=GID`)	**Signed filter (`hash`)**
Filters shown in the URL	no (stored server-side)	yes (visible)
Tamper-proof	yes	yes
Expire / revoke / use-limit	yes	no
Needs server-side minting	yes	no (any key holder signs)

Signed Filters (`hash`) — *optional*

> **Almost everyone should use Hidden Links instead** —
> they hide the filters *and* support expiry/revocation. Signed filters exist only for
> the one thing hidden links can't do: sign a URL **without** minting a server-side row.

Reach for this only when you must generate signed URLs **statelessly** — e.g. a separate app building report URLs programmatically, very high volume where a table row per link is impractical, or several servers sharing one key. Then add an `*api_key*` row to `report_writer_security.csv` (any text in the `path` column; keep it secret):

```
logical_name,path,security
*api_key*,a-long-fully-random-key,
```

With a key set, **every run URL must carry** `&hash=phash$(filter1..5 concatenated + the key)` (filters in `filter1, filter2, …` order); an unsigned or wrongly-signed request is rejected. The bundled `make_filter_hash.spsrc` computes the `&hash=` for a set of filters. Use a long, fully random key — it's never typed, so length is free, and a short one can be brute-forced offline by anyone who captures a single signed URL.

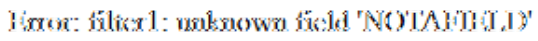
Error Handling

The run endpoint always returns an HTML fragment; on a problem it returns a styled **error panel** (and plays a short error tone) instead of a report:

Condition	**What you get**
Missing `format`	"Error: Missing format name" with usage hint.
Unknown `format`	"Error: Format not found".
Outdated definition file	"Incompatible Format" with an option to delete it.
Table cannot be opened	"Error: Cannot open table" with the OS message.

Condition	What you get
Bad/unknown `filterN`	A styled **error panel** naming the filter and reason — e.g. `filter1: unknown field 'X'`, `filter1: unknown operator 'X'`, `filter1: missing value`, `filter1: 'abc' is not a number for QUANTITY`. The report is withheld until the filter is corrected, so a typo never silently returns the wrong rows.
Missing `hash` (a key is configured)	"Error: missing &hash= (this report requires a signed URL)".
Wrong `hash`	"Error: invalid &hash= for these filter values".
Bad `api` link	"Error: This link is not valid / has been revoked / has expired / has reached its use limit."

A bad `filterN` is reported the same way, naming the filter and the reason:



To detect an error from a host page, fetch the fragment and check for `class="error-panel"`, or look for the absence of `class="report-table"`.

Styling the Embedded Report

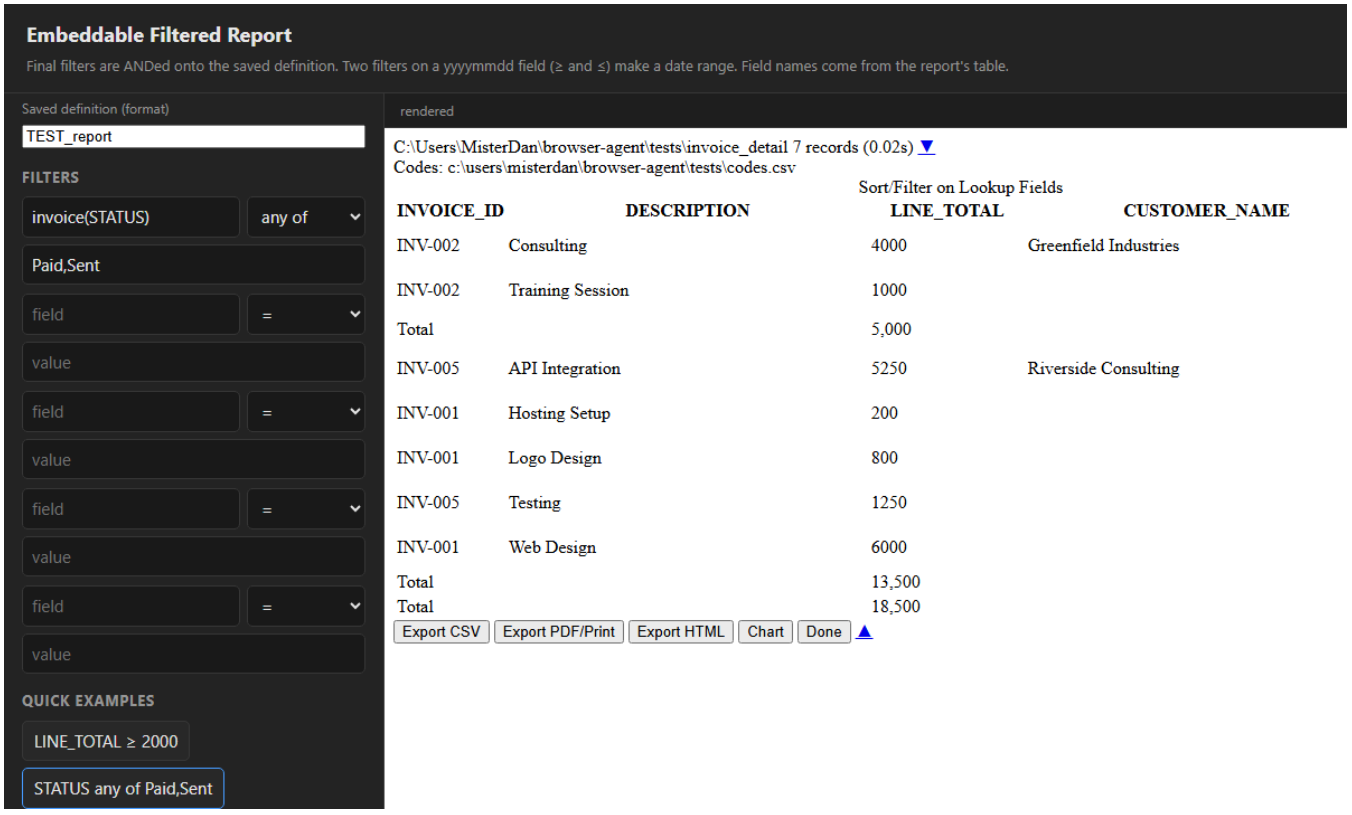
The returned fragment is **semantic HTML**: a `<table class="report-table">` inside `<div class="report-table-wrap">`, with a `<h2 class="report-title">`, colored subtotal/total rows (inline colors), and a small toolbar (Export CSV / PDF/Print / HTML / Chart / Done). Out of the box it renders with the browser's default table styling plus those inline stat colors — readable, but not themed.

For pixel-matched Report Writer styling inside the frame, serve the report's stylesheet (`report_writer.css`) within the framed document, or wrap the embed in your own host page and apply your site's table CSS to `.report-table`. The data, totals, lookups, and code translations are all present regardless of styling.

Live Demo Page

A self-serve demo, `report_filter_demo.html`, lets you pick a definition, build FIELD-OP-VALUE filters (with field suggestions, operator dropdowns, and quick-example buttons), and watch the embedded report update live — while showing the exact embed URL to copy into your own page:

```
/report_filter_demo.html
```



Reference Card

```
Endpoint:  /scripts/spiis.dll/report/report_writer.html
Always:    action=run
Required:  format=<saved definition name>            (case-insensitive on Windows)
           plain name   -> @spreports\name.spreport  (folder beside the program)
           logical:name -> <path>\name.spreport      (logical must be listed in
report_writer_security.csv)
Filters:   filter1..filter5                          (ANDed onto the definition)
           inline:  FIELD<op>VALUE                    (comparisons only)
           pipe:    FIELD|OP|VALUE[|VALUE2]           any op; required for the multi-char
ops
           ops:     = <> > < >= <=
                    contains  startswith  endswith
                    range -> |LOW|HIGH (either side blank = open)
                    anyof -> comma list, matches any
           field:   BARE | table(FIELD) | lookup(FIELD)
           values:  numbers bare; text auto-quoted; bad/unknown filter -> error panel
           NO url-encoding needed for operators/|/commas; encode space/&/#/+ in values
```

```
Hidden:    api=GID    opaque link -- format/filters stored server-side, hidden &
           untamperable; expirable/revocable/use-limited. Create by pasting a run
           URL into make_api_url.spsrc (prints + clipboards ?action=run&api=GID)
Signing:   advanced/stateless alternative to Hidden -- &hash= via *api_key* (see Signed
Filters)
Export:    csv=auto | csv=<name>                     (data rows only)
           pdf=1                                      (opens print dialog)
Returns:   HTTP 200, Content-Type: text/html         (error-panel body on failure)
Embed:     <iframe src="…?action=run&format=…&filter1=…"></iframe>
```