



User Guide

Enterprise reporting, distribution, and governed AI - product documentation

Version 6.0.0.0

Contents

User Guide

What Reportworq is	5
Sign in and take the tour	8
Back, Forward, and shareable links	14
Choose the source workbook and worksheets	16
What bursting is and when you need it	20
Distribution overview	23
The Office add-ins	26
Functions overview: which one do I use	28
Run a job	35
Track activity and run history	40
Contribution (write-back) overview	44
Key concepts and the mental model	47
Your first report, end to end	50
Roles and what you can do	54
Global parameters	59
Replicate output from your source system	62
Address book and contacts	66
Excel: build and refresh reports	70
General report and layout functions	75
Schedule a job	87
Run estimates and expected completion	91
Author a contribution campaign	96
The report lifecycle	106
Item properties	109
Parameterize a report	112
Burst sets	118
Output formats	123
Excel: find and insert functions	130
The AI insight function	132
Manage scheduled content	136
View and download results	139
Fill in and submit a form	142
Collect data with an input workbook	149
Store files in the workspace	152
Parameters reference	154

Assign recipients to bursts	164
Name and organize output files	168
Excel: charts and maps	173
Workday Adaptive functions	185
When a job fails	194
Contribution functions	198
Advanced parameters: variables and cascading	205
Destinations	208
IBM Planning Analytics functions	213
Review a run's performance	217
Profile and stress-test a campaign	223
Worksheet output options	232
Email messages and templates	235
Relational database functions	240
Find when a report got slower	245
Scale and size a contribution campaign	250
Job options: collation and splitting	260
Combine emails, collate, and zip	262
Anaplan functions	267
Save and Save As	270
The Workspace Inbox	272
PowerPoint reporting	278
Charts and maps	287
Create an AI Profile for a report	291

User Guide

What Reportworq is

explanation

report author

user

Reportworq is an enterprise reporting and distribution platform. Planning systems are excellent at collecting budgets, forecasts, and actuals and at building financial plans. What they generally do not do is get that information into the hands of the people who need it to run the business. Reportworq fills that gap.

Reportworq turns the reports finance already builds into governed, trusted assets that refresh, personalize, and deliver themselves, so your team gets its analysis time back.

The problem it solves

In a small organization, reporting is manageable: someone builds a spreadsheet, saves a version, and emails it or drops it on a shared drive. As an organization grows, that breaks down. Picture a retailer with a thousand stores. Every store manager needs a personalized view, every regional manager needs different information, and executives need consolidated numbers. Finance ends up creating, personalizing, and distributing thousands of reports, and if a single number changes the whole process starts again. That is operational overhead, not value creation.

Reportworq automates that entire process. An author builds one report, and the platform refreshes the data, personalizes each copy, and delivers every output on time, without anyone assembling and emailing files by hand.

Meet people where they already are

The people who need the information are often not the people who work inside the planning system. Sales leaders, department managers, executives, HR leaders, operational teams, and even external partners need the information, not a seat in the planning platform. Buying more planning-system licenses is rarely the right answer.

So Reportworq delivers instead of inviting. It sends finished reports to email, Microsoft Teams, Slack, SharePoint, network folders, and more, as PDF, Excel, PowerPoint, CSV, or rich HTML. Rather than forcing people into another application, Reportworq meets them where they already work.

Not finance-only

Finance is the primary use case because finance already has standardized, repeatable reporting, but Reportworq is not limited to financial data. Any standardized report that is

built once and needs to go out on a regular cadence, operational reporting, HR, sales operations, executive flash reports, fits the same pattern. Reportworq is strongest in financial reporting today and extends naturally into any workflow where enterprise information must move reliably between systems and people.

Coming full circle: contribution

Distribution sends information out. **Contribution** brings it back.

A **contribution campaign** turns a report into an input form. You send it to the people who own the numbers, a cost-center manager, a regional lead, a department head, and they fill in their part: a forecast, a headcount plan, a commentary field. Reportworq collects what they submit, routes it for approval where you need one, and **writes the approved values back into the planning system**.

That closes the loop. The same governed report that goes out as a PDF can come back as a form, so the round trip from "here are your numbers" to "here is my plan" happens in one platform, with the same security and the same audit trail. Nobody is reconciling emailed spreadsheets. See [Contribution overview](#).

The AI layer

Reportworq connects your people, and their AI chat clients, to the reports you have already built and curated.

That is a deliberately narrow claim, and the narrowness is the point. Reportworq's AI does **not** try to reconstruct your business model, invent data queries nobody can validate, answer questions it has no grounds to answer, or burn tokens exploring your data warehouse. It sits **on top of reports that finance has already approved**.

What you get from that:

- **Answers from trusted content.** The AI reads finished, reviewed report output, so what it says traces back to a report someone signed off on.
- **Security that already exists.** A user reaches through AI exactly what they could reach in the product. There is no second permission model to keep in step.
- **Governance and auditing.** AI access is logged like everything else, so you can see who asked what and which content answered.
- **Steering wheels.** Authors add context to a report telling the AI what it means and how to talk about it, which is how you keep answers from drifting or hallucinating. See [Create an AI Profile for a report](#).

For the full picture, see [AI in Reportworq](#).

Governed throughout

Reportworq is a *governed* platform. Reports live in a shared workspace with security, scheduling, run history, and audit logs, so an organization can see who can run what, when a report last ran, and where its outputs went. That governance is what makes both of the above safe: contribution writes back through it, and AI answers from within it.

Where to go next

- Learn the one idea that unlocks the product in [Key concepts and the mental model](#).
- Sign in and get oriented in [Sign in and take the tour](#).
- Build a report end to end in [Your first report, end to end](#).

Sign in and take the tour

how-to

report author

user

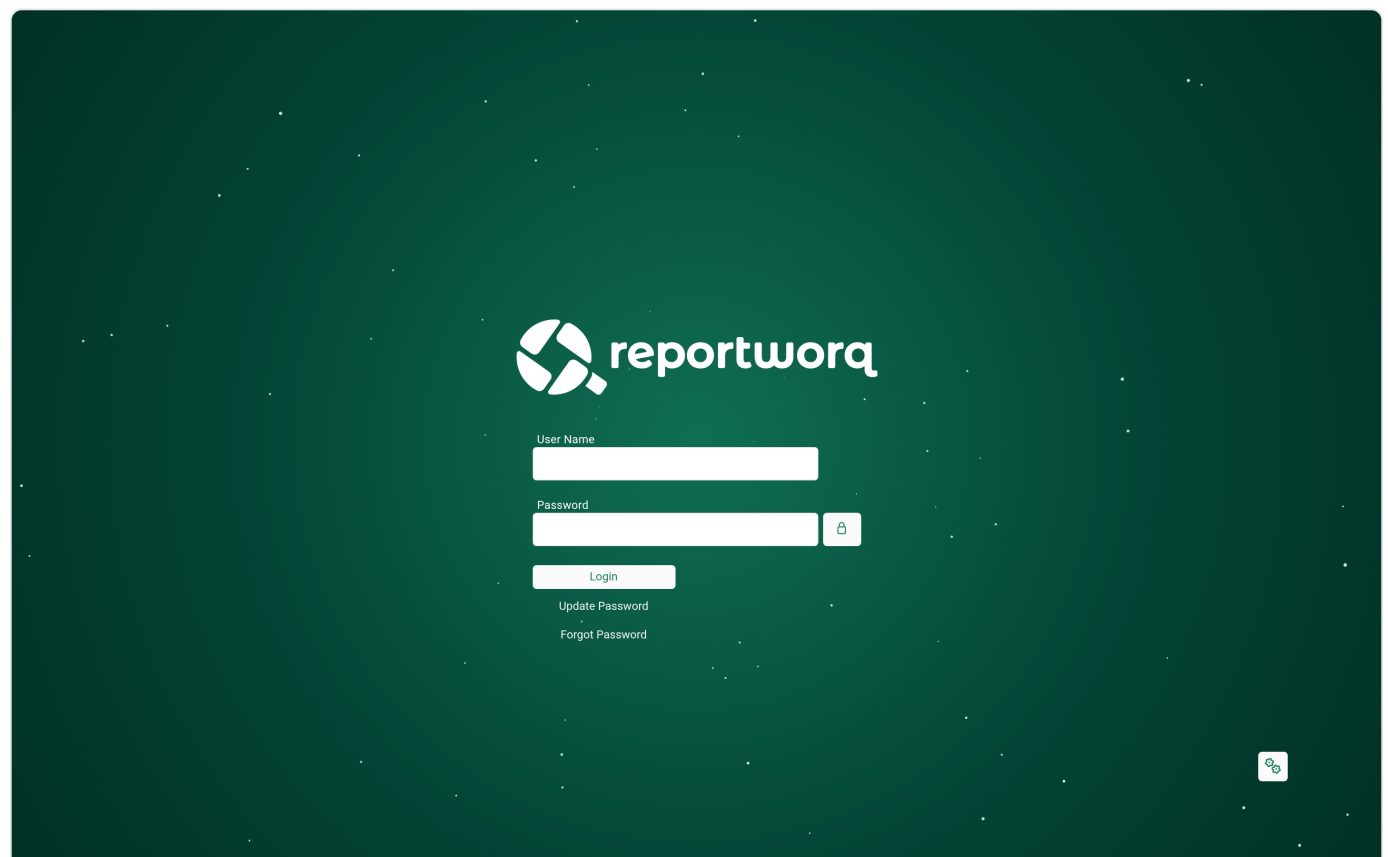
This page walks you through signing in to Reportworq and finding your way around the home screen. By the end you will know how to sign in, where the main tools sit, and how to sign out.

Before you start

- A Reportworq account with access to at least one workspace. An administrator provisions this; signing in alone grants nothing until your role and permissions are set.
- The web address of your Reportworq server.

Reportworq uses **Native** username and password sign-in by default. If your organization has enabled single sign-on, the sign-in screen redirects to your corporate identity provider instead of showing the username and password fields, and your role comes from your identity-provider group membership. Setting up SSO and recovering from a lockout are administrator tasks; see [Sign-in and SSO](#).

Sign in



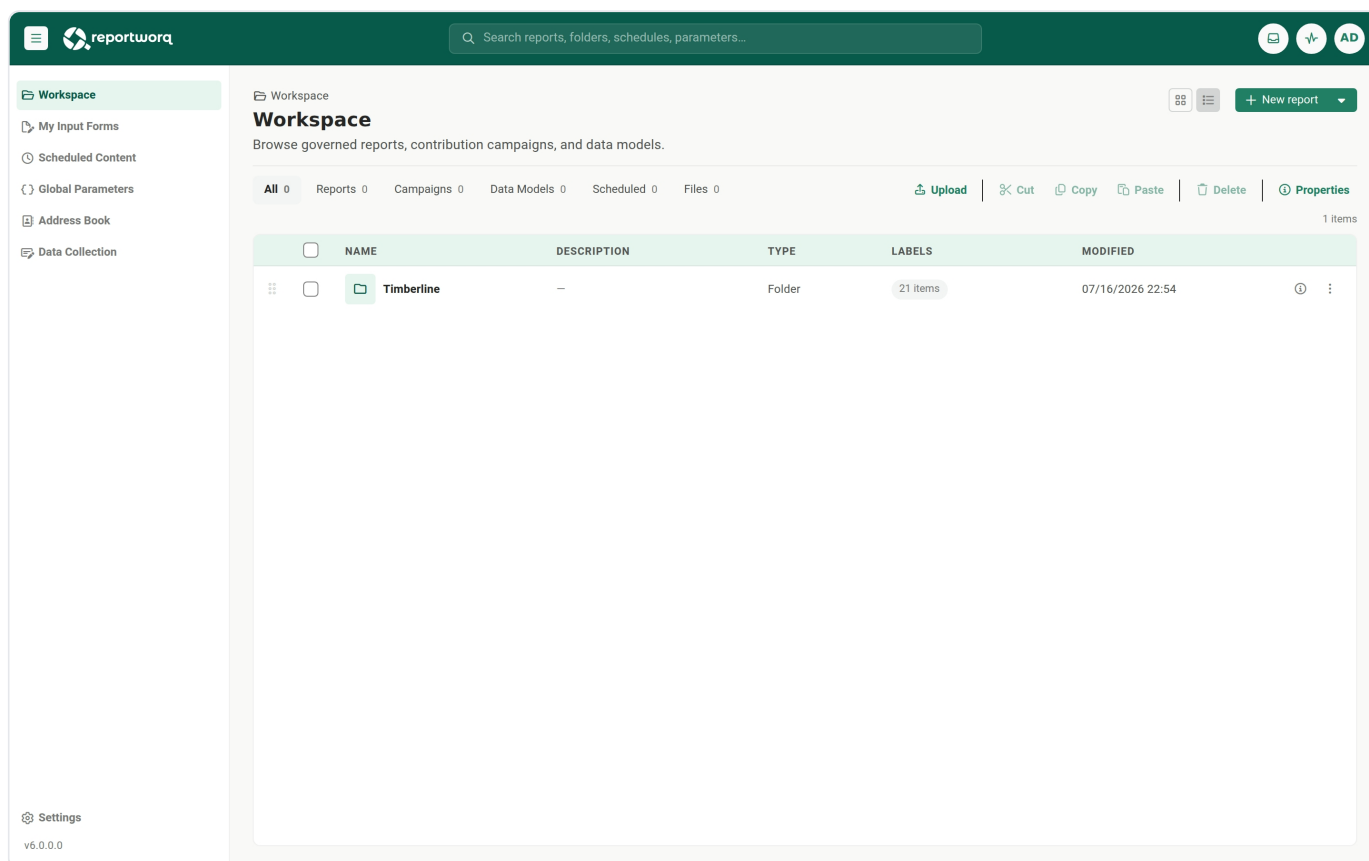
1. Open your Reportworq web address in a browser. The native sign-in screen appears.
2. In the username field, enter your account email.
3. In the password field, enter your password.
4. Select **Login**.

A brief transition animation plays while the app loads, and then the Workspace opens.

If you select **Login** with an empty username field, Reportworq reports *"A user name is required to authenticate."* and stays on the screen. If your account is new, you are prompted to set an initial password before the app opens. A **Forgot Password** link appears only when your administrator has configured an email server.

Take the tour

After sign-in, Reportworq lands on the **Workspace**, described on screen as *"Browse governed reports, contribution campaigns, and data models."* Every signed-in screen shares the same frame.



What you see is scoped to your role. This tour describes the author and administrator shell; a view-only User sees a trimmed, read-only version with no **New** button and no header search. See [Roles and what you can do](#).

Stop 1: the frame

Find each part before you touch anything:

- **The header** runs across the top and holds the **global search** box, the activity and notification indicators, and the **account menu** at the far right, which opens **Help**, **Support**, **Settings**, and **Logout**.
- **The side rail** down the left lists the primary areas, **Workspace**, **Scheduled Content**, **Global Parameters**, **Address Book**, and **Data Collection**, with **Settings** in the footer above the server version badge (for example `v6.0.0.0`).
- **The content pane** in the middle is where screens open in place.

Try it: select **Scheduled Content** in the side rail, then **Workspace** to come back. Screens open in the content pane without ever leaving the shell.

Reportworq keeps the browser address bar in step with the screen you are on, so your browser's **Back** and **Forward** buttons work, refreshing keeps you where you are, and you can bookmark or share a link to a specific place: a folder, a report's run history, a settings tab, even a step in the job editor. See [Back, Forward, and shareable links](#).

Stop 2: the content library

The Workspace browser has two regions:

1. A **folder card grid** at the top. Select a folder card to open it and see what is inside.
2. An **item table** below it, with the columns **Name**, **Description**, **Last Run**, **Schedule**, and **Outputs**. **Last Run** and **Outputs** tell you at a glance whether something ran and what it produced.

Try it: open a folder from the card grid, then scan the **Last Run** column.

reportworq

Search reports, folders, schedules, parameters...

AD

Workspace

My Input Forms

Scheduled Content

Global Parameters

Address Book

Data Collection

Settings

v6.0.0.0

Workspace > Timberline > Financial Reports

Financial Reports

Reports, campaigns and data models in this folder.

All 12

Reports 8

Campaigns 2

Data Models 2

Scheduled 0

Files 0

Upload

Cut

Copy

Paste

Delete

Properties

13 items

	NAME	DESCRIPTION	TYPE	LABELS	MODIFIED	
	Source Files	—	Folder	9 items	07/17/2026 20:47	
	Board Packet	Executive board PPT deck	Report		07/17/2026 20:48	
	Board Packet with Division Slides	Board Deck with slice by division f...	Report		07/17/2026 20:51	
	Commission Statements	Commission statements for most ...	Report		07/16/2026 22:10	
	Divisional Packet	This is a high level board deck cre...	Report		07/17/2026 20:52	
	Financial Statements	Income, Balance Sheet, and Cash ...	Report		07/23/2026 20:48	
	Financial Statements - SharePoint	—	Report		07/20/2026 21:03	
	Sales Compensation	—	Campaign	Running	07/27/2026 15:40	
	Sales Journals	—	Data Model	SQL Provider - Transaction...	07/20/2026 17:26	
	Sales Rep Scorecard	This report is to provide managem...	Report		07/17/2026 20:54	
	Segment & Channel Budget Input	Budget input tool by segment and ...	Campaign	Running	07/24/2026 17:57	
	SQLDB	—	Data Model	SQL Provider - Timberline J...	07/27/2026 12:19	
	Timberline Sales	—				

To act on an item, use its row: the inline **edit**, **run**, and **info** actions, or the overflow ... menu for the full set (**Edit**, **Run**, **View**, **Run history**, **Properties**, **Rename**, **Duplicate**, **Delete**).

Stop 3: narrow the list

Try it: select the **Reports** filter chip.

The item table narrows to reports only, and the chip's count badge shows how many the current folder holds. Select **All** to return to the full list.

The chips are **All**, **Reports**, **Campaigns**, **Data Models**, and **Scheduled**. Filter to **Reports** to hide campaigns and data models in a large library, or to **Scheduled** to see what runs on a schedule. The **layout toggle** switches between **Card** and **Grid**, and your choice is remembered per account across sessions.

Stop 4: long lists

Lists that grow with your data are paged, so a folder with thousands of entries stays responsive.

- A pager sits under the list with the current page, the page controls, and the count shown out of the total.

- **100 rows** per page is the default. Choose **100**, **500**, **1000**, or **All**. **All** renders every row, which is what you want before printing or copying a whole list out, and is slower on a large one.
- Searching, filtering, or sorting applies to the **whole list**, not only the page on screen, and the pager reports the filtered total.

The same pager appears on the address book, scheduled content, a job's parameters, global variables, burst sets, the report picker, My Input Forms, and a campaign's forms list. Two lists behave differently by design: a **campaign dashboard** pages its form cards in smaller batches with no page-size selector, and a data model's **Fields** tab is not paged at all, because fields are reordered by dragging and a drag cannot cross a page boundary.

Stop 5: search

When you know an item's name, search beats browsing.

Try it:

1. Select the **global search** box in the header (placeholder *"Search reports, folders, schedules, parameters..."*).
2. Type at least **2 characters**. Results appear across reports, folders, schedules, parameters, accounts, groups, contacts, outputs, and global parameters.
3. Use the up and down arrows to move the selection, **Enter** or a click to open the highlighted result, and **Esc** to close the panel.

Results are scoped to your access, so you only ever see what you are permitted to. Search finds a *known* item fast; to browse *by type*, use the filter chips instead.

Stop 6: create something

To add an item, use the **New** split-button in the browser toolbar. Its caret menu offers **New Folder**, **New Report**, **New Campaign**, and **New Data Model**. Selecting a type prompts for a name; new reports, campaigns, and data models are created in place and appear in the list. Opening one with **Edit** takes you into its editor.

The **New** button and the authoring toolbar appear only for authors and administrators.

Sign out

When you are finished:

- Open the **account menu** in the header, then select **Logout**.

Reportworq ends your session and returns to the sign-in screen.

Next steps

- Build your first report in [Your first report, end to end](#).
- See how a report moves from idea to delivery in [The report lifecycle](#).

Back, Forward, and shareable links

[how-to](#)[report author](#)[administrator](#)[contributor](#)[user](#)

Reportworq keeps the browser address bar in step with the screen you are on. That means the browser's own **Back** and **Forward** buttons work, any screen can be **bookmarked** or **refreshed** without losing your place, and you can **share a link** that opens a colleague on exactly the same screen. This is on by default; there is nothing to enable.

Move Back and Forward

Use the browser **Back** and **Forward** buttons to step through the screens you have visited - the Workspace, Activity, Scheduled content, Settings sections, an open editor, and so on. An open editor is kept alive as you move, so stepping **Back** into a report you were editing keeps your unsaved changes.

When a dialog or a side panel is open, **Back** closes it first, rather than navigating the page underneath - so one press does what you expect.

Bookmark or refresh a screen

Because the address bar tracks the current screen, you can:

- **Bookmark** a screen and return to it later - a Settings tab, a workspace folder, a report's run history, or the Job Editor at a particular step.
- **Refresh** the page and stay exactly where you were, instead of being sent back to the start.

Share a link to a screen

Copy the address bar and paste it to a colleague. When they open it (in a fresh tab, even after a browser restart), it lands on the same surface - for example a folder in the workspace, a specific report's history, the Activity list filtered to failures, or a Settings tab. If the link points at something their account is not allowed to see, Reportworq opens their home screen instead of exposing it.

Open a folder in a new tab

Breadcrumbs are real links. **Middle-click** or **Ctrl-click** (or **⌘-click**) a breadcrumb to open that folder in a new browser tab; a plain click navigates in place as usual.

Notes and limits

- This applies to the main Reportworq web application. It does not change how external integrations (the API or the MCP and Copilot surfaces) work, since those do not use the browser shell.

Choose the source workbook and worksheets

AUTH-01

how-to

report author

Every Reportworq report renders one or more **source files**, Excel workbooks or PowerPoint templates, into output. The **Reports** step of the Job Editor is where you choose those source files and pick which of their **worksheets** the report outputs, and in what order. This is the first thing you do when building a report.

The worked example is a **monthly P&L pack** built from one workbook that holds a summary tab, detail tabs, and a hidden lookup tab.

Before you begin

- Open the report in the **Job Editor** and select the **Reports** step. A new report starts empty and shows "0 reports, 0 sheets."
- A **report provider** (a configured source-file location) with at least one source file must exist. If none is configured, **Add a Report** tells you there are no report providers. Ask your administrator to set one up, or upload a file from within the dialog.

What you can use as a source

Excel is the report. `.xlsx`, `.xls`, `.xlsm`, and `.xlsb` are all accepted, and this is the file Reportworq refreshes, parameterizes, and renders.

You can also attach PDF, Word, and PowerPoint files, and it is worth knowing what they do before you plan around them. A `.pdf`, `.docx`, or `.pptx` added as a source is **appended into the job's PDF output**. Reportworq does not edit or refresh them, and the Excel render pass ignores them entirely, so they only appear when the job produces a PDF. That makes them right for fixed front matter, a cover page, standing notes, or terms and conditions that travel with every pack. It makes them wrong for anything that needs to change with the data.

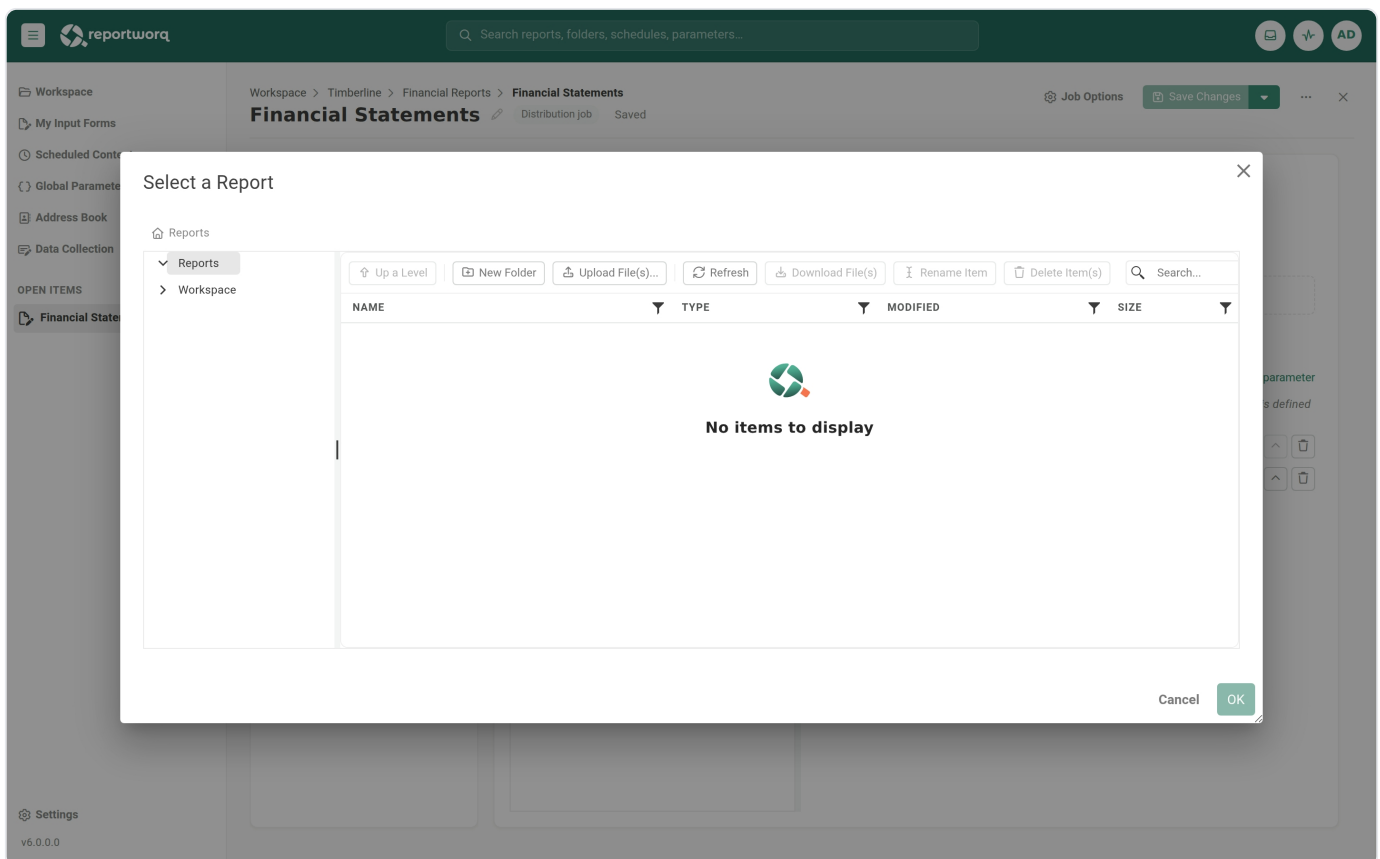
Encrypted Excel workbooks are not supported as sources. Remove the encryption before you add the file.

Add a source workbook

1. On the **Reports** step, select **Add a Report**.

2. In the **Select a Report** dialog, browse the report provider's folder tree and select your source file, for example `Monthly_PnL.xlsx`. The dialog shows the same folders, names, and permissions as the Workspace file browser.
3. Select **OK**.

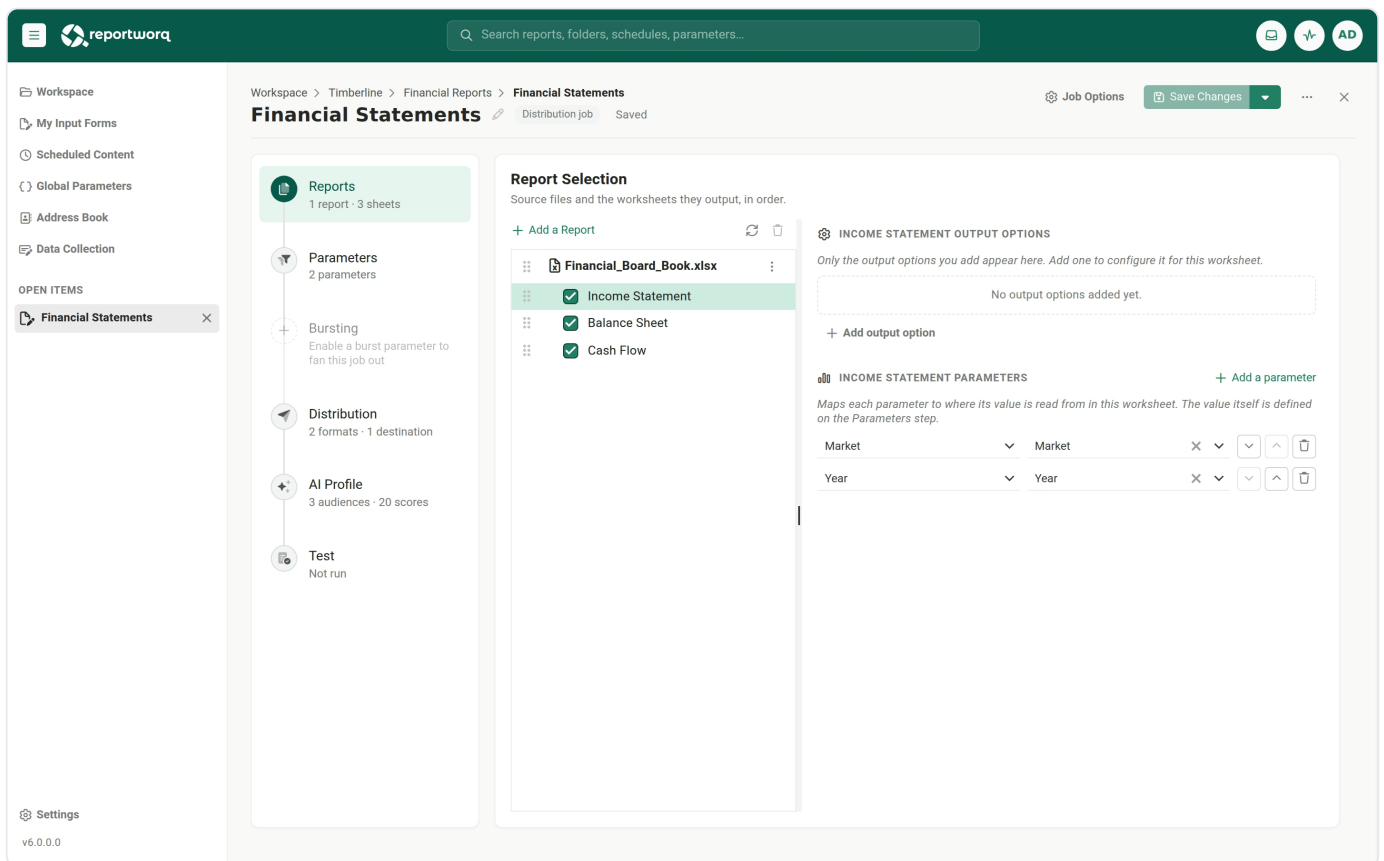
To bring in a file that is not yet in the provider, use **Upload File(s)...** inside the dialog. Upload honors the provider's allowed file extensions. Uploading a `.zip` prompts you to **extract** it (which recreates the archive's folder structure) or store it as a single file.



Choose the worksheets to output

Once a source is added, its worksheets appear as a checklist.

1. Select the checkbox next to each worksheet you want in the output. Clear the checkbox for any worksheet you want to leave out.
2. Drag the worksheets into the order you want them rendered. Included worksheets are output in list order.



A report can hold **multiple source files**. Add each one with **Add a Report**; each contributes its selected worksheets to the output in sequence, so you can combine tabs from two workbooks into a single report.

Each included worksheet also exposes an **Output Options** panel and a **Parameters** mapping panel. See [Worksheet output options](#) and [Parameterize a report](#).

The most common mistake here is mapping a parameter to the wrong worksheet. A workbook with several similar tabs makes it easy to map `Region` onto the summary sheet when you meant the detail sheet. It fails quietly: the report generates, the parameter resolves, and the numbers are simply not filtered the way you intended. After mapping, run the Test step and check the value landed on the sheet you meant. See [Parameterize a report](#).

Manage a source file

Each source file has a context (the `...`) menu with these actions:

- **Open, Download, Rename, Properties, Remove.**
- **Relink** re-points the source to a different file, for example after the file has moved.

Notes and limits

- **Excluded worksheets still calculate.** An unchecked worksheet is removed from the output before distribution, but it stays in the workbook during calculation. It can still feed the visible tabs through formulas, lookups, commentary, imagery, or Cell Value message and file-name variables. Use a hidden calc or lookup tab to drive visible content without shipping the tab itself. In short: compute here, do not ship.
- **Encrypted Excel workbooks are not supported** as sources. Remove workbook encryption before adding the file.
- **File and folder names in the report provider cannot contain** `< > : " / \ | ? *`. Uploading, renaming, or creating a folder with one of those characters is refused with a message naming them, on every host operating system. Rename the file before you upload it.
- **Cross-workbook formulas resolve only when the workbooks are co-located.** A formula in an Excel source can reference another workbook only if that other workbook sits in the **same folder as the source file, within the same Network Share report provider**. Co-locate referenced workbooks, or the references break at render time.
- **Map the replicating parameter to every worksheet that must replicate.** This is the rule behind copying worksheets as a group. When a parameter replicates a worksheet (East, Central, West) and a second worksheet references it, map that **same parameter on both worksheets**, even if the second one does not need to display the value. Reportworq then copies the related worksheets together and rewrites intra-workbook cross-sheet formula references, and pivot-table data sources, to point at each copy's own sibling rather than the original. Map it on only one worksheet and the other will not replicate with it, so every copy keeps reading the original. Copying as a group is on by default; an administrator can toggle it at **Settings, Configuration, Reporting, Advanced Features, Enable copying worksheets as a group**. Worked through in [Parameterize a report](#).

Going deeper. To make one report serve many periods, entities, and recipients, add parameters. See [Parameterize a report](#).

What bursting is and when you need it

explanation

report author

Bursting is the outcome of one report producing many personalized outputs: a P&L per cost center, an expense report per department, a statutory pack per legal entity. You build one source template, and Reportworq produces and delivers the whole set.

There are two different mechanisms that produce that outcome, and confusing them is the single biggest source of wasted effort in this part of the product.

Three words, kept straight

Term	What it means
Bursting	The outcome. One report, many outputs. It is a result, not a screen.
Replication	A parameter that holds more than one value fans the output out, one output per value. You set the parameter's replication mode on the Parameters step and you are done. The Bursting step is not involved at all.
Burst set	One row on the Bursting step. Each row carries its own value for each burst parameter and, optionally, a recipient. The Bursting step is hidden until a parameter is flagged Burstable .

Throughout the guides, output fanned out by a parameter's own values is called **replication**, and **burst set** always means the Bursting-step feature.

The distinction people miss: the **Burstable** flag is not what makes output multiply. A parameter that already holds several values and has a replication mode is producing many outputs whether or not it is flagged Burstable. The flag does exactly one thing: it unlocks the **Bursting** step. If you are not building burst sets, leave it off.

Which one you need

Use replication when:

- The values you want to fan out on already exist somewhere you can read them, such as a Planning Analytics subset or MDX query, a SQL query, a Workday Adaptive dimension, an Excel range, or a typed Text List.
- Every output is built the same way and differs only by that value.
- The list should stay current on its own as the business changes, with no second list to maintain.

This is the common case, and it is where most jobs should land. See [Replicate output from your source system](#).

Use burst sets when:

- You need **per-output control**: a different value, a different query, or a different recipient for individual outputs rather than the same treatment for every one.
- You have a **fixed, curated list** with no home in a source system, such as a roster of stores or an external distribution list.
- The list of outputs is easiest to maintain as a spreadsheet or a standalone query that the business already keeps.

See [Burst sets](#).

What a burst set adds

A burst set is worth the extra step when the outputs are not all built the same way. The Bursting grid gives you one row per output and one column per burst parameter, so each output can carry its own values.

The part that saves the most work is the **default**. Every cell starts by inheriting the value already set on the **Parameters** step, and it shows as **Default** with that value. You only edit the cells that need to differ, which become **overrides**.

That matters most when a burst parameter is query-driven. If the parameter is a Planning Analytics MDX query or a SQL query, the query you enter once on the Parameters step is the default across every burst set. A grid of 30 sets needs that query entered once, and you override only the two or three sets that need a different query or a fixed value. At run time each set applies only the values it actually carries, and every cell left at Default falls through to the job parameter.

How the two fit together

Replication and burst sets are separate axes, and a job can use both.

- Replication is set per parameter with its **replication mode**. **One report per item** produces a separate output file per value. **One page per item** produces extra worksheets inside one output instead of more files. **No replication** declares that the parameter is a single value.
- Burst sets sit outside that. Reportworq works through the burst sets first, applies each set's values on top of the job's parameters, and then replicates within that set. A job with four burst sets whose parameters do not replicate produces four outputs. If a parameter still replicates inside each set, each set produces one

output per value.

- When no burst sets exist, the job behaves as a single implicit set, which is why replication works perfectly well on a job that never opens the Bursting step.

Recipients are optional

Neither mechanism requires you to name a recipient. An output that has no recipient is still produced and still delivered, using the job's own destinations, such as a fixed network folder or a static email list.

When outputs do need to reach different people, the recipient can come from several places: an attribute alongside the value in your source system, a value on a worksheet in the report, a **Contact** column in an Excel or SQL burst source, or a contact picked by hand on a burst set. The address book is one convenient way to hold that information, not the required one. See [Assign recipients to bursts](#).

What each path needs

- Both paths need at least one parameter on the job. A parameterized report is the prerequisite for every form of bursting. See [Parameterize a report](#).
- For replication, the parameter's type can hold more than one value. See [Parameters reference](#) for the full type catalog and the replication modes.
- For burst sets, decide where the list of sets will come from: the grid, a workbook, or a query.

Going deeper. A parameter can reference another parameter's value inside its own query and repeat that query per item, which produces cascading one-report-per-item fan-out. See [Advanced parameters](#).

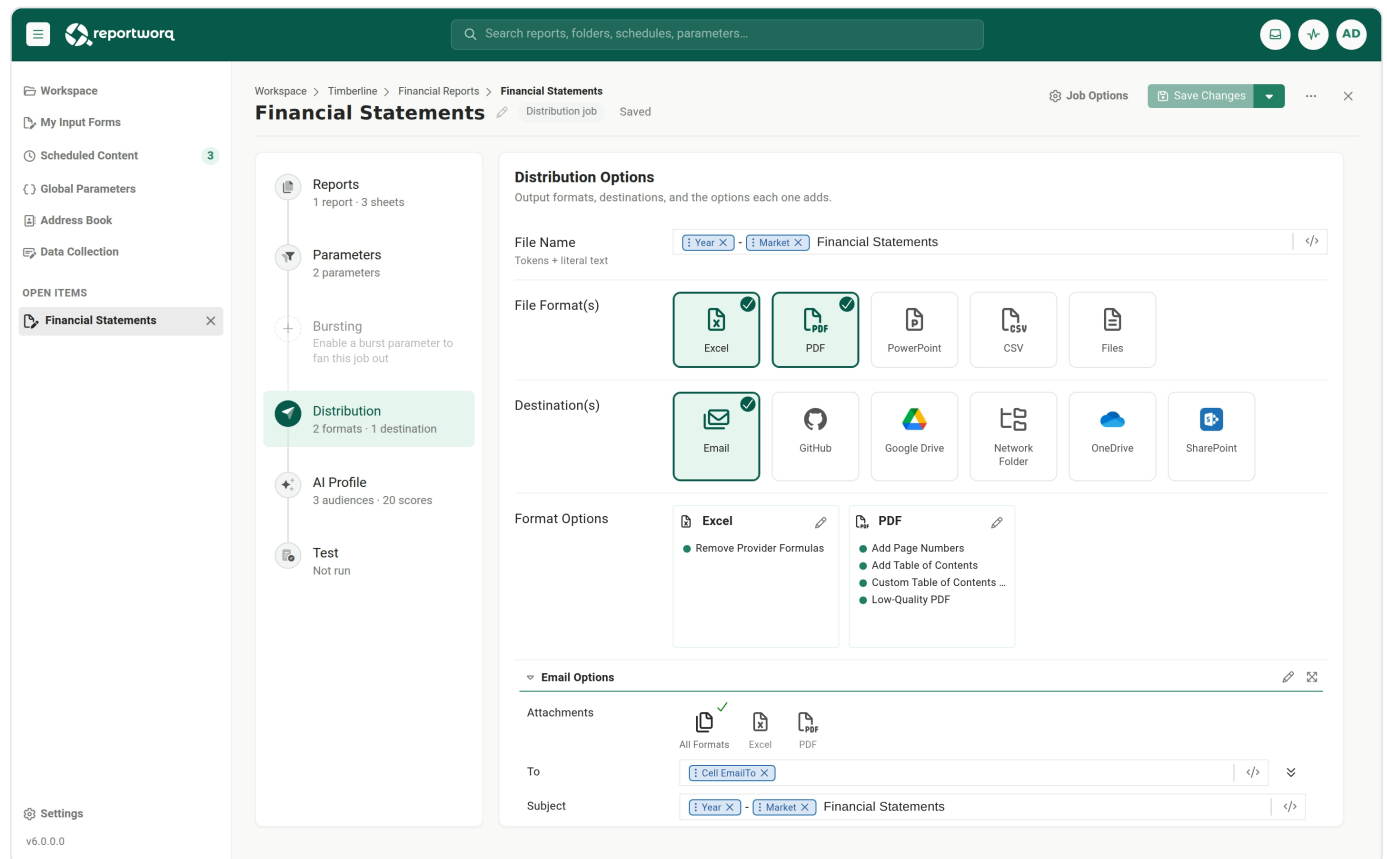
Distribution overview

DIST-02

explanation

report author

Authoring builds a report; **distribution** delivers it. The **Distribution** step of the Job Editor is where you decide three things: what **output formats** the report generates, how each file is **named and organized**, and which **destinations** receive it. It is the step that turns a refreshed Excel template into the personalized email, the archived PDF, or the Slack post your recipients actually see.



The mental model: formats times destinations

A single job is not locked to one file type or one place. On the Distribution step you select output formats as a set of tiles, and destinations as a second set of tiles, and Reportworq produces the **cross-product**: every selected format is generated once, then each destination delivers the formats you route to it.

That means one job can, in a single run:

- render the report to **Excel and PDF and PowerPoint** at the same time, and
- email the PDF to controllers, drop the Excel on a network folder, and post a Slack notification,

without you building a separate job per format or per place. A common misconception is that each output format needs its own job. It does not. Select as many format tiles as you need, and route them where they belong. See [Output formats](#).

The three decisions on the Distribution step

Decision	What you set	Where it is documented
What files	The output-format tiles (Excel, PDF, PowerPoint, CSV, Files) and each format's options.	Output formats
What they are called and where they land	The file-name pill editor and each destination's folder or path field.	Name and organize output files
Where they go	The destination tiles (Email, Network Folder, SharePoint, cloud drives, Slack, Teams, SFTP, IBM Planning Analytics).	Destinations

Two more sidebars shape how the finished files are packaged and worded:

- **Message templates** let you write a branded, dynamic subject and body once and reuse it, so you are not editing the email by hand every week. The same editor is where an **AI-generated insight** is added to a message, and where recipient addresses can be read from the workbook instead of the address book. See [Email messages and templates](#).
- **Combine, collate, and zip** group multiple outputs together, for example one email that carries a recipient's whole pack instead of five separate messages. See [Combine emails, collate, and zip](#).

When files are generated versus when they are sent

Generating a file and sending it are two separate stages. A report can render successfully and still send nothing, by design, in a few cases worth knowing up front:

- **Hold for Delivery** runs the job and previews the files but does **not** send them. The held files are kept so you can review and release them later from Activity. This is the safe way to check output before it reaches anyone.
- A destination only delivers the formats you routed to it. If a destination is left with no formats, it sends nothing even though generation succeeded.
- An **exception check** can act as a send gate for the whole report. If the report has an exception cell and its condition was not met, the send is suppressed for that run.

These are covered where you configure them, in [Destinations](#) and (for exception checks) in [Worksheet output options](#).

Where this fits in the report lifecycle

Distribution comes after you have chosen the source workbook and worksheets and parameterized the report, and before, or alongside, scheduling it to run on a cadence. Bursting decides *how many* copies a run produces and *who* each one is for; distribution decides *what each copy is* and *where it goes*.

Going deeper. To send a different copy to each store, entity, or recipient from one template, pair distribution with bursting. See [What bursting is and when you need it](#).

The Office add-ins

[explanation](#)[report author](#)

Reportworq loads a task pane inside the Office applications your authors already live in. There are two, and they do different jobs.

Add-in	What you do with it
Excel	Pull governed data into a worksheet layout you control, refresh it on demand, find and insert Reportworq worksheet functions, and build refreshable charts and maps.
PowerPoint	Link slide shapes to report objects so a deck fills with live data on every run. The linking procedure lives with the rest of the PowerPoint story, in PowerPoint reporting .

What the add-ins are for, and what they are not

The add-ins are for **content**: getting the right data into the right shape on a sheet or a slide.

They are **not** where a job is assembled. Bursting, distribution, scheduling, and everything about who receives what stays in the web Job Editor. The usual division of labor is that an author builds and refreshes the content in Office, then goes to the web to decide how it is produced and where it goes.

Shared prerequisites

Both add-ins need the same three things:

- **HTTPS on the Reportworq server.** This is a hard requirement for every Microsoft 365 feature, not a recommendation. Over plain HTTP the add-in sign-in fails, and it fails quietly. See [System requirements](#).
- **The add-in installed in Office.** Install from **Home > Add-ins**, search for **Reportworq**, then **Add > Continue > Allow and Continue**. On Excel 2016, or anywhere users cannot install add-ins themselves, an administrator deploys it centrally. See [Deploy the Office add-in](#).
- **A signed-in licensed account.** You sign in every time you use the add-in. The prompt accepts a non-default server URL and port (the Windows default is 8600).

Where to go next

- [Excel: build and refresh reports](#)

- [Excel: find and insert functions](#)
- [Excel: charts and maps](#)
- [PowerPoint reporting](#), which covers linking slide shapes to report objects in the PowerPoint task pane
- [Functions overview](#), for what the worksheet functions do once they are in the sheet

Functions overview: which one do I use

[reference](#)[report author](#)

Reportworq extends Excel with a set of custom worksheet functions you place in a source workbook. They are **inert in desktop Excel** and are evaluated by the Reportworq engine when a job runs, or, for write-back functions, when a contribution form is submitted. Rather than returning a value to the cell the way a native Excel function does, most of them perform an action, import a query result, reshape the layout, set a worksheet property, or write data back to a source, and then clear or replace the formula cell.

Every function has two names that resolve to the same thing:

- a **canonical dotted name**, for example `RW.Tools.Sort` or `RW.Relational.Export`, and
- a **legacy short alias**, for example `RWSORT` or `RWSQL`.

New workbooks should use the canonical form; the aliases are kept for compatibility with older templates. The one exception is `ANAPLANINPUT`, which has no dotted name.

The **AI*** functions are Workday Adaptive, not generative AI

This is the single most common misreading of the function surface, so read it before you go further.

The worksheet functions whose aliases begin with **AI** (`AIINPUT`, `AIEXTRACT`, `AIFILTER`, `AIOPTION`, `AITIER`, `AIMODELEDINPUT`, `AIMODELEDEXPORT`) are the **Workday Adaptive Planning** connector functions, canonically `RW.Adaptive.*`. Here "AI" is short for **Adaptive Insights**, the former name of Workday Adaptive Planning. They have **nothing to do with artificial intelligence or large language models**.

The only generative worksheet function is `RW.Tools.AIInsight` (`RWAIINSIGHT`). See [The AI insight function](#).

Two ways to put source data on a sheet

Worksheet functions are not the only way to get data into a workbook. The Reportworq add-in does it too. The two routes reach the same destination and differ mainly in how you work.

The add-in is the visual route. You build a report in the companion pane, and the pane is where you craft and manipulate both the data being imported and how it renders: which fields come across, how they are grouped, sorted, aggregated, and filtered, how the columns are formatted, and where on the sheet the block lands. The report keeps a spec inside the workbook, so you can reopen it and reconfigure it without rewriting anything. See [Excel: build and refresh reports](#).

A worksheet function is the plain route. You drop a query straight into a cell. There is no interface and no stored report spec. The formula is the whole definition, and at run time the result lands where the formula sat. That plainness is the appeal: what sits in the cell is what produced the data, and the layout around it is entirely yours to design.

Neither one is the advanced option. Both accept the same `{{parameter}}` tokens, so a query that varies by region, entity, or period can be expressed either way, and both produce a live report that refreshes on every run. Some people prefer the keyboard, and some prefer the mouse.

Two practical points usually settle the choice:

- When the output is naturally a table that grows and shrinks with the data, the add-in's list report carries the grouping, sorting, aggregates, and formatting for you.
- When the output is a designed page with numbers in specific places, a formatted financial statement or a board pack, functions bring the data to your layout instead of an imported block dictating it.

Plenty of real workbooks use both: a formatted statement built from functions, with a supporting detail tab imported through the add-in.

Complete function index

Every function, with a direct link to its full reference entry.

General report and layout: `RW.Tools.*`

Reference page: [General report and layout functions](#)

Function	Alias	What it does
RW.Tools.Sort	RWSORT	Sorts a range or table by up to three keys, optionally keeping only the top or bottom N rows
RW.Tools.Suppress	RWSUPPRESS , RWZERO	Hides or deletes rows or columns when a condition is true. Defaults to delete
RW.Tools.Group	RWGROUP	Builds Excel outline groups from repeated row or column identifiers
RW.Tools.BlankRow	RWBLANKROW	Inserts blank rows above or below the formula row
RW.Tools.PageBreak	RWPAGEBREAK	Inserts a horizontal or vertical page break
RW.Tools.Resize	RWRESIZE	Auto-fits a row height or column width, merged cells included
RW.Tools.RemoveFormula	RWREMOVEFORMULA	Replaces formulas with their values
RW.Tools.SheetName	RWSHEETNAME	Sets the worksheet output name from a formula
RW.Tools.SheetColor	RWSHEETCOLOR	Sets the worksheet tab color from a formula
RW.Tools.SheetSuppress	RWSHEETSUPPRESS	Suppresses a worksheet from a formula

IBM Planning Analytics

Reference page: [IBM Planning Analytics functions](#)

Function	Alias	What it does
RW.PlanningAnalytics.MDX	RWMDX	Runs a TM1 MDX query and writes the cellset into a range
Native TM1 functions		DBRW , DBS , SUBNM , VIEW , TM1RPT* , DIMIX and the element and dimension helpers

Relational database

Reference page: [Relational database functions](#)

Function	Alias	What it does
RW.Relational.Export	RWSQL , RWRELATIONALEXPORT	Runs a SQL query and imports the result into a range or named table
RW.Relational.Update	RWSQLUPDATE	Updates the row matching a key column and value
RW.Relational.Upsert	RWSQLUPSERT	Same as Update, but inserts when the key is missing

Workday Adaptive Planning

Reference page: [Workday Adaptive functions](#)

Function	Alias	What it does
RW.Adaptive.Extract	AIEXTRACT	Builds and imports a custom Adaptive report
RW.Adaptive.Tier	AITIER	Defines one row, column or filter axis for an Extract
RW.Adaptive.Option	AIOPTION	Sets a named report option that Extract reads
RW.Adaptive.Filter	AIFILTER	Stores a reusable dimension filter
RW.Adaptive.Input	AIINPUT	Marks a cell for upload to Adaptive on submit
RW.Adaptive.ModeledInput	AIMODELEDINPUT	Updates one field on a modeled Adaptive sheet
RW.Adaptive.ModeledExport	AIMODELEDEXPORT	Exports modeled Adaptive data into a range or table

Anaplan

Reference page: [Anaplan functions](#)

Function	Alias	What it does
ANAPLANINPUT	<i>(no dotted name)</i>	Marks a cell as an Anaplan write-back input

AI

Reference page: [The AI insight function](#)

Function	Alias	What it does
RW.Tools.AIInsight	RWAIINSIGHT	Calls a configured AI provider and writes the response into a TextBox or cell

Contribution: `RW.Contribution.*`

Reference page: [Contribution functions](#)

Function	Alias	What it does
<code>RW.Contribution.Input</code>	<code>RW.INPUT</code>	Declares the editable range and its dimensional context
<code>RW.Contribution.Check</code>	<code>RW.CHECK</code>	Adds a validation rule, blocking or advisory
<code>RW.Contribution.Checkbox</code>	<code>RW.CHECKBOX</code>	Adds a checkbox bound to a target cell
<code>RW.Contribution.List</code>	<code>RW.LIST</code>	Adds a dropdown of allowed values
<code>RW.Contribution.Filter</code>	<code>RW.FILTER</code>	Adds a filter control and its value list

Which function do I use

Goal	Function
Bring TM1 or Planning Analytics data into a sheet with MDX	RW.PlanningAnalytics.MDX
Read a single TM1 cell or slice with the classic functions	DBRW , DBR , SUBNM , VIEW , TM1RPT*
Write a value back to a TM1 cube cell	DBS , DBSW , DBSS
Bring SQL data into a sheet or named table	RW.Relational.Export
Write back to SQL where the row already exists	RW.Relational.Update
Write back to SQL, creating the row if missing	RW.Relational.Upsert
Build a custom Workday Adaptive report	RW.Adaptive.Extract with Tier and Option
Write back to Workday Adaptive	RW.Adaptive.Input or ModeledInput
Write back to Anaplan	ANAPLANINPUT
Sort a data block, optionally keeping top or bottom N	RW.Tools.Sort
Hide or delete rows and columns on a condition	RW.Tools.Suppress
Group repeated rows or columns into an outline	RW.Tools.Group
Add spacing rows in a report that grows at run time	RW.Tools.BlankRow
Control print pagination	RW.Tools.PageBreak
Auto-fit a row or column around runtime content	RW.Tools.Resize
Freeze calculated results before distribution	RW.Tools.RemoveFormula
Set a sheet's name, tab color or suppression from a formula	SheetName, SheetColor, SheetSuppress
Build an interactive contribution input form	RW.Contribution.*
Embed an AI-generated summary or commentary	RW.Tools.AIInsight

How the functions run

- **Evaluated by Reportworq, not Excel.** The formulas do nothing in desktop Excel. They are recognized and executed by the Reportworq engine at the right phase.
- **Action, not value.** After a successful run, most functions clear the formula cell. The data importers instead set the cell text to the query that produced the data.

- **A fixed layout pipeline.** During report generation the data importers run first, then the layout functions in a set order: `RW.Tools.Sort` , then `RW.Tools.BlankRow` , then `RW.Tools.Group` , then `RW.Tools.Suppress` , then `RW.Tools.PageBreak` and `RW.Tools.RemoveFormula` . `RW.Tools.AIInsight` runs after import, sort and suppression. Ordering matters when functions interact: a top-N sort must run before rows are suppressed.
- **Phase separation.** Reporting, layout and data-read functions run during **report output**. Write-back functions (`RW.Relational.Update` and `Upsert` , the `RW.Adaptive.*` inputs, `ANAPLANINPUT` , and the `RW.Contribution.*` inputs) are processed during **input submission**, not the output pass.

The quiet failures worth knowing

Most function mistakes do not produce a formula error. These are the ones that fail silently, each covered on its reference page:

- `RW.Tools.Suppress` **deletes** by default when you probably meant hide.
- `RW.Tools.Group` treats the **first** matching row as the header and leaves it outside the group.
- `RW.Relational.Update` and `Upsert` **ignore an odd trailing argument**, so a mistyped column pair writes nothing.
- `RW.Adaptive.Option` **ignores an unknown option name**, so a typo looks like the option had no effect.
- `RW.Adaptive.ModeledExport` splits `levels` on `|` , not commas.
- `RW.Tools.AIInsight` **silently drops `dataRange`** when its worksheet cannot be found, and only warns when the destination TextBox does not exist.
- `ANAPLANINPUT` trailing tuple arguments **suppress the breadcrumb** that the write-back actually depends on.
- The three sheet functions apply **nothing at all** if two of the same kind appear on one worksheet.

Related pages

- [General report and layout functions](#)
- [IBM Planning Analytics functions](#)
- [Relational database functions](#)
- [Workday Adaptive functions](#)
- [Anaplan functions](#)
- [The AI insight function](#)
- [Contribution functions](#)

Run a job

how-to

report author

administrator

Run Jobs is the on-demand way to run a report right now, instead of waiting for a schedule. You pick one or more jobs, choose which of their outputs to produce, optionally override parameters for this run, set a run mode and logging, and launch. The run is queued and you land on **Activity**, where you can watch it finish.

When to use it. Any time a report has to run outside its schedule: an ad-hoc request mid-month, a re-run after you fixed a template, a review-gated delivery you want to check before it sends, or a diagnostic run with extra logging. For recurring, unattended runs, set up a [schedule](#) instead.

Before you begin

- You need at least one runnable job with resolvable outputs.
- For outputs to actually be delivered, the job needs a configured distribution. **Hold for Delivery** still produces output without delivering it.

Pick the jobs and outputs

1. Open **Run Jobs**. The header reads "Pick jobs to run now, review the outputs they produce, then set run options." A left rail steps you through **Selection**, **Parameters**, and **Options**.
2. On the **Selection** step, use the **Available Jobs** tree on the left. Search or expand folders, then select the checkbox next to each job you want. Folders roll up their jobs, so checking a folder selects everything in it.
3. Each job you check is added live to **Outputs to run** on the right and its outputs are resolved. A card shows "{included} of {total} outputs included" per job.
4. Include or exclude individual outputs with their row checkboxes, or use **Include all** / **Exclude all** on a job's card. A **Burst** chip on a card means that job resolves to more than one output.
5. To start over, select **Clear**.

You can run at least one output across the selected jobs, so keep at least one output included or the run control stays disabled.

Workspace > Run Jobs Cancel Run Jobs

Run Jobs
Pick jobs to run now, review the outputs they produce, then set run options.

Selection
1 job - 13 of 13 outputs

Parameters
3 parameters - none overridden

Options
Production

AVAILABLE JOBS

Search...

- Examples 3 jobs
- Timberline 12 jobs
 - Financial Reports 12 jobs
 - Source Files empty
 - Board Packet
 - Board Packet with Division Slides
 - Commission Statements
 - Data Validation Check
 - Divisional Packet
 - Divisional Packet - Burst Set Example
 - Financial Statements
 - Financial Statements - Job Failure
 - Financial Statements - output examples
 - Financial Statements - SharePoint
 - ☒ Sales Rep Scorecard
 - Timberline Sales Information

OUTPUTS TO RUN 13 of 13 included Clear

☒ **Sales Rep Scorecard** **BURST** ^ ×
Financial Reports - 13 of 13 outputs included

☒	#1	#1 E0005 2026	Pdf
☒	#2	#1 E0006 2026	Pdf
☒	#3	#1 E0007 2026	Pdf
☒	#4	#1 E0008 2026	Pdf
☒	#5	#1 E0009 2026	Pdf
☒	#6	#1 E0010 2026	Pdf
☒	#7	#1 E0011 2026	Pdf
☒	#8	#1 E0012 2026	Pdf

Include all Exclude all

When to run several jobs together. Selecting multiple jobs and running them in one go runs them in the same execution process, which is what lets Reportworq group them into a single message when **Combine email messages when possible** is turned on. See [Combine emails, collate, and zip](#).

Override parameters for this run (optional)

1. Select the **Parameters** step. It is available only when a selected job exposes an overridable parameter.
2. The step shows the parameters shared across your selected jobs. Enter a value to apply it to every selected job that uses that parameter for this run only.
3. Leave a value blank to use each job's own default. Each override can be reset back to the default.

Choose the run mode and options

1. Select the **Options** step. A recap reads "Ready to run {N} outputs across {M} jobs."
2. Choose a **run mode**:
 - **Production** (the default) generates the outputs and distributes them.
 - **Hold for Delivery** generates the outputs but pauses distribution so you can review them first. You release or discard the held run later from [Activity](#).
3. Under **Logging**, turn on the extra detail you need. Both switches are off by default, and a routine run needs neither.
 - **Verbose logging** captures detailed step-by-step execution logs, so the run's log records each step it worked through rather than only the point where it stopped.
 - **Trace logging** adds high-volume diagnostics, including the HTTP traces of the calls the run made to its data sources. It is available only while **Verbose logging** is on, and Reportworq turns it off again after this run.
4. Set notifications:
 - **Notification email** receives a run summary when the run completes. It is pre-filled from your saved settings.
 - **Notify on failures only** stays quiet on success and emails only if something fails.

Workspace > Run Jobs Cancel Run Jobs

Run Jobs

Pick jobs to run now, review the outputs they produce, then set run options.

Selection
1 job · 13 of 13 outputs

Parameters
3 parameters · none overridden

Options
Production

✔ **Ready to run 13 outputs across 1 jobs.** Excluded outputs will be skipped. Set how this run behaves below.

▶ RUN MODE

☒ **Production**
Generate outputs and distribute to recipients.

☐ **Hold for Delivery**
Generate outputs but pause distribution for review.

📄 LOGGING

Verbose logging
Capture detailed step-by-step execution logs. ☐

Trace logging
High-volume diagnostics - auto-disables after this run. ☐

✉ NOTIFICATION

📎 A run summary is sent here when the run completes.

Notify on failures only
Stay quiet on success; email only if something fails. ☐

When to turn logging up

Leave both switches off for routine production runs. There are two situations that call for **Verbose logging**.

- **A job keeps failing while you are still setting it up.** A verbose run records every step, so the run's log shows where the job stopped and what it was doing at the time, instead of only that it failed. Turn **Trace logging** on as well when the failure looks like a data-source problem: the HTTP traces exist only for a run that had Trace logging on, and cannot be recovered afterward.
- **Support has asked you for logs.** Re-run the job with **Verbose logging** on, and **Trace logging** too if you were asked for it, so the run's diagnostics archive carries the detail support needs. Then send that run's archive from its diagnostics, see [When a job fails](#).

Logging is set on the run you are launching, not on the job. The next run of the same job records its normal level of detail unless you turn logging on again.

Launch the run

1. Select **Run Jobs** in the page header. The run is queued and you are taken to **Activity**.
2. If your workspace has the split-batch entitlement, the run control is a split button. Its menu adds **Run Individually**, which queues one run per job in its own process. Combined email messages cannot span jobs run individually.

Notes and limits

- There is no "Test Mode." The only run modes are **Production** and **Hold for Delivery**.
- A run is gated on at least one included output, so include an output before you launch.
- Canceling a running job (from Activity) gives it 60 seconds to finish and clean up before it is stopped.

Going deeper. Watch the run, release a held delivery, or review what it produced in [Track activity and run history](#). If a run fails, see [When a job fails](#).

Track activity and run history

[how-to](#)[report author](#)[administrator](#)

Reportworq gives you two monitoring views. **Activity** is the live, cross-report view of what is running, what is held for review, and what recently ran. **Run History** is the per-report record of every run of one report. Use Activity to watch and manage runs in flight; use Run History to trace one report over time.

When to use it. Open Activity during and just after a run: to watch a month-end burst move through, release a delivery you were reviewing, cancel a stuck job, or spot a failure. Open a report's Run History when you want the full record of that one report, its past outputs, and its retention.

Watch runs on Activity

Activity presents one grid, "Running tasks, held deliveries, and recent run history," in a fixed order: **Running**, then **Hold**, then **History**.

1. Open **Activity** (you also land here after launching a [Run Jobs](#) run).
2. Each row shows a **Category** chip (**Running**, **Hold**, or **History**), the job **Name**, **Start** time, **Elapsed**, a **Status**, and row actions.
3. Two more columns appear when they carry information. **Workspace** appears on an installation with more than one workspace, naming the workspace each run belongs to. **Server** appears when runs record the node that executed them, which happens on a load-balanced installation. On a single-workspace, single-node installation both stay hidden rather than repeating a constant on every row.
4. The grid refreshes in place as runs progress and finish. **Refresh** re-queries on demand.
5. Filter with the **Status** filter (**All**, **Running**, **Held**, **Warnings**, **Failures**) and the **date range** filter (**Today**, **Last 7 Days**, **This Month**, **This Year**, **All**). The date range scopes History rows only.

Activity

Activity

Running tasks, held deliveries, and recent run history.

Activity 27

AllRunningHeldWarningsFailures

Last 7 Days

	CATEGORY	NAME	START	ELAPSED	STATUS	
☐	History	Sales Compensation	07/27/2026 18:40	00:00:07	Success	
☐	History	Sales Compensation	07/27/2026 16:04	00:00:09	Success	
☐	History	Sales Compensation	07/27/2026 16:01	00:00:11	Success	
☐	History	Sales Compensation	07/27/2026 15:55	00:00:11	Success	
☐	History	Sales Compensation	07/27/2026 15:43	00:00:05	Success	
☐	History	Sales Compensation	07/27/2026 15:41	00:00:15	Success	
☐	History	Sales Compensation	07/27/2026 15:37	00:00:09	Success	
☐	History	Timberline Sales Information	07/27/2026 12:05	00:02:42	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:55	00:00:04	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:52	00:00:04	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:51	00:00:04	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:45	00:00:04	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:42	00:00:04	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:39	00:00:04	Success	
☐	History	Segment & Channel Budget Input	07/24/2026 17:36	00:00:04	Success	

<<< < > >>

Page 1 of 1

> >>

100 items per page

1 - 27 of 27 items

Activity

Activity

Running tasks, held deliveries, and recent run history.

Activity 7

AllRunningHeldWarningsFailures

Last 7 Days

	CATEGORY	NAME	START	ELAPSED	STATUS	
☐	History	Job Failure Example	07/31/2026 20:26	00:00:00	Failure	
☐	History	Job Failure Example	07/31/2026 20:25	00:00:00	Failure	
☐	History	Financial Statements - Job Failure	07/31/2026 20:20	00:00:13	Failure	
☐	History	Financial Statements - output examples	07/31/2026 20:17	00:00:09	Warnings	
☐	History	Divisional Packet	07/31/2026 19:59	00:00:39	Success	
☐	History	Board Packet with Division Slides	07/30/2026 16:30	00:00:19	Success	
☐	History	Data Validation Check	07/29/2026 21:11	00:00:02	Success	

<<< < > >>

Page 1 of 1

> >>

100 items per page

1 - 7 of 7 items

Read run success from a completed **History** row's status (Success, Warnings, or Failure) or from Run History, not from a Running row's progress text. Mid-run text can include the word "completed" while the job is still going.

Release or cancel a held delivery

A run made in **Hold for Delivery** generates its outputs but pauses distribution, so you can review before anything is sent. Held runs appear in the **Hold** category.

1. Review the held output first. Open [the Report Viewer](#) or the run's **Details** to see what was produced.
2. On the held row, select **Release All** to distribute the held outputs and clear the hold. Confirm when prompted. Releasing ships what was already produced; it does not re-run the report.
3. To keep the output internal and not send it, open the row's ... menu and select **Cancel Hold**, which clears the hold without distributing.

Either action moves the run into **History**.

Cancel a running job

1. On a **Running** row in your current workspace, select **Cancel** (stop).
2. Confirm in the **Cancel Job** dialog.
3. The run is given 60 seconds to finish and clean up before it is stopped, so a canceled row can stay Running briefly before it clears.

Other row actions

- **Details** (History or Hold rows) opens the run's detail, which hosts Details and Diagnostics. See [When a job fails](#).
- **Open in Workspace** (in the ... menu) jumps to that report's Run History.
- **Delete** (in the ... menu) removes the run from the list. It asks you to confirm first, and it cannot be undone from Activity.

Review a report's Run History

Run History is the full record for one report, newest first.

1. Open it from a job row's ... menu (**Run history**), or from a run's **Open in Workspace** in Activity.

2. The run list shows each run's **Status** and **Started** time, three performance columns (**Elapsed vs baseline**, **Trend**, and **Peak mem**), then **Outputs**, **Formats**, and **Delivery**. Filter with the **All / Success / Warning / Failed** pills, each carrying a count. For what the performance columns compare a run against, see [Review a run's performance](#).
3. Open a run to inspect its outputs, per-file **downloads**, **Distribution history**, and any **AI Insights**. A failed run shows an error card with **Run again** and **Send logs to support**.
4. **Pin** a run to keep it regardless of the retention policy, for example a final period-close run. The retention note on the list states how long runs are kept. **Delete run** removes the run from the history list; it asks you to confirm, and warns that the run's stored output and execution log go with it.
5. If a run you know has finished is not listed, select **Refresh** in the ... menu. Refresh rebuilds the list from the stored history rather than re-reading what is already in memory, so a completed run whose finish notification went missing appears immediately. The same applies to **Refresh** in [the Report Viewer](#).

Notes and limits

- Activity is for authors and administrators. Users do not have access to Activity.
- Activity is the cross-report live view; Run History is the per-report record. To read or download a run's rendered output, use [the Report Viewer](#).
- **Deleting a run cannot be undone from Activity or Run History.** Neither screen has a restore action. The deleted run's stored output and execution log stay on the server until a background purge removes them, so a run deleted by mistake can sometimes still be recovered, but only as a Reportworq support task and only before the purge runs. Contact Reportworq support promptly if you need one back. To keep a run indefinitely instead, **Pin** it. See [Output retention](#).

Going deeper. To read, download, or email a run's outputs, see [View and download results](#). If a run failed or warned, see [When a job fails](#). To find out why a run was slow, see [Review a run's performance](#).

Contribution (write-back) overview

explanation

Most of Reportworq sends information out: a report becomes a PDF, an Excel file, or a slide deck, and it lands in an inbox or a folder. **Contribution runs the other way.** It hands people an input form, collects what they enter, and writes those values back into a planning system.

The target is a real plan, not a mailbox. Collected numbers post back to an IBM Planning Analytics (TM1) cube, a SQL table, or a Workday Adaptive model, so the loop closes where the plan actually lives.

The mental model

Three ideas carry the whole feature:

- A **campaign** is the object an author builds. It defines the input forms, who fills them, who approves them, the notifications sent along the way, and the export that pushes collected data to a target system.
- An **input form** is what a contributor sees. It is an Excel-based form rendered in the browser: input cells to type into, checkboxes and dropdowns to pick from, and validation that flags problems before submission. A planning toolbar above the grid carries the input tools and bulk operators, and a collapsible Details dock on the right holds the workflow status, history, and per-cell validation. See [Fill in and submit a form](#).
- An **export** is what moves data out of the campaign. A campaign holds the values contributors enter in its own runtime. Nothing reaches the planning system until an export runs, gathers the collected values, and writes them to the configured target: a Planning Analytics cube, a database table, or a file. An export runs on a trigger (commonly when a form is approved), on a schedule, or by hand from the campaign dashboard.

A campaign is built on the same report machinery as any other Reportworq job, source worksheets, parameters, and optional bursting, so an author who can build a report can build a campaign. The difference is that the worksheets carry contribution formulas that turn plain cells into editable, validated inputs.

Contribution and data collection are not the same process

Data collection is a separate Reportworq feature for collecting input workbooks, and a campaign does not run through it. Both write to planning systems, both are covered by the same license, and they are documented next to each other for that reason, but they are two distinct processes that do not blend today. They differ at every step of a round.

Step	Contribution campaign	Data collection
What the person sees	An assigned input form in the browser, in My Input Forms	An Excel workbook they received or downloaded
How input comes back	Submitted in the browser, with validation and optional approvals	Returned as a workbook, by upload or to a monitored mailbox
What writes to the plan	The campaign's export, mapping collected fields to the target	The data-collection engine, running the workbook's hidden write-back formulas
Where it is run and watched	The campaign dashboard	Configuration ► Data Collection, and the write-back activity view

Data collection stands on its own merits. Sending a workbook out and letting an inbox collect it back is still the right answer when contributors work offline, when the round is small enough that assignments and approvals are overhead, or when an existing workbook already carries its write-back formulas. Reach for a campaign when you want assigned ownership, in-form validation, approvals, and a dashboard that shows who has not submitted. See [Data collection processing](#).

How a cycle runs

1. An **author** builds a campaign: designs the form worksheets, generates and assigns forms, sets notifications, and configures the export target. See [Author a contribution campaign](#).
2. **Generate Forms** turns the definition into concrete forms, one for every output the report definition would produce. A definition that repeats per value, whether through a parameter set to **One report per item** or through burst sets, fans into many forms, for example one budget form per cost center, so each contributor sees only their slice. A definition that produces a single report produces a single form.
3. Before the round opens to real people, the author **measures and sizes it**: how much memory a form costs, how many contributors a server carries, and whether the experience holds up under load. See [Scale and size a contribution campaign](#).

4. **Contributors** fill their forms in My Input Forms, clear any validation errors, and submit. See [Fill in and submit a form](#).
5. If approvals are on, an **approver** reviews each submitted form and approves or rejects it. One approval closes a form, regardless of how many approvers are assigned.
6. An **export** pushes the collected data to the target system, on a trigger (for example on approval, or on a schedule) or run by hand from the dashboard.

Who does what

Role	Where they work	What they do
Campaign author	The campaign editor	Designs forms, assignments, notifications, and exports.
Contributor	My Input Forms	Fills and submits assigned forms.
Approver	My Input Forms	Reviews submitted forms; approves or rejects.
Administrator	Configuration	Licenses contribution, keeps the write-back targets reachable, and runs data collection processing.

Contribution is not a separate account type. Anyone who can sign in, a User, an Author, or an Administrator, can be assigned as a contributor or approver on a Contribution-licensed installation, and **My Input Forms** simply appears in their side rail alongside everything else they already see.

When to use contribution

Reach for a campaign when you need to **gather data from people and put it back into a plan**: a budget submission round, a headcount input round, a driver-based forecast, or any cycle where owners enter numbers you then consolidate. If you only need to send a report out, a standard report job is the right object, not a campaign.

Write-back requires the licensed **Contribution** capability, which includes Data Collection, to post data to a target. Without it you can still author campaigns and collect input, but the export cannot write to the plan. See [Data collection processing](#).

Going deeper. The formulas that define input cells, validation, and controls live in the form worksheets. See [Contribution functions](#).

Key concepts and the mental model

explanation

report author

user

Most Reportworq concepts fall into place once one idea clicks. That idea is the single most important thing a new user has to get.

A report is a template that makes outputs

Do not think of a report as an output. Think of it as a **template that creates outputs**.

A finished PDF or a sent email is a *result*. In Reportworq, the report is the thing that *produces* those results, over and over. It is a reusable asset with levers built in: parameters, variables, queries, formatting, messaging, and distribution rules. Pull those levers, and one report can generate hundreds or thousands of personalized outputs.

Different parts of the same source workbook do different jobs. Some parts exist to build the output file. Some exist only to build a rich email message. Others make distribution decisions, and others exist to collect data back into the planning system. The report becomes a specification for how information should flow through the organization.

This is where new users most often get stuck: they keep thinking of a report as a document rather than a reusable system. Once you build one solid master template that Reportworq can slice, personalize, and repeat, you stop maintaining a separate file per audience, and that is the moment the product pays off.

The terms that follow

Everything else is a name for one of the levers on that template.

- **Report (distribution job).** The unit an author builds in the **Job Editor**. Internally it is a *distribution job*: a single runnable, schedulable object that binds source content, parameters, optional bursting, output formats, and delivery targets together. "Report" and "job" are the same thing seen from two angles, the author's asset and the thing that runs.
- **Source file and worksheet.** The Excel workbook (or PowerPoint template) the report renders, and the specific sheets it outputs, in order. The source file is where your existing Excel work lives, Reportworq automates it rather than replacing it. See [Choose the source workbook and worksheets](#).
- **Parameter.** A named input that drives the report. Change a parameter such as *period*, *store*, or *cost center*, and the formulas in the source workbook recalculate

to that value. Parameterizing a report is what lets one template serve every audience. See [Parameterize a report](#).

- **Bursting, replication, and burst sets.** **Bursting** is the outcome: many personalized outputs from one job, one per value, each going to its own recipient. One cost-center job becomes one packet per cost-center owner.

Two words describe *how* you get there, and they are not interchangeable.

Replication is a parameter with several values fanning the output out by itself; set the parameter's replication mode and you are done. A **burst set** is the separate **Bursting** step, unlocked by flagging a parameter **Burstable**, which gives you per-file control over the values and the recipients.

You do not need Burstable to produce many outputs. A multi-value parameter is already replicating. Flag Burstable only when you want the Bursting step itself. See [What bursting is and when you need it](#).

- **Distribution.** The output formats a job produces (PDF, Excel, PowerPoint, CSV, and more) and the destinations it delivers them to (email, network folder, SharePoint, Slack, Teams, and others). One job can produce several formats to several places at once. See [Distribution overview](#).
- **Workspace.** The governed content space a report lives in, its own folder tree, items, contacts, and security. The Workspace is also the home screen you browse. A server can host more than one workspace.

Most organizations need only one. Folder-level security handles separating teams inside a single workspace, which is simpler to run than several. The clearest reason to add a second is **credentials**: a workspace holds its own data-source connections, so when two groups must reach the same system under *different* credentials, giving each its own workspace is the way to do it. Splitting purely to separate content is usually better served by folders.

- **Contribution.** The reverse direction. A **contribution campaign** turns a report into an input form, hands it to the people who own the numbers, collects what they enter, routes it for approval where you need one, and writes the approved values **back** to the planning system. The pieces are the **campaign** (the round of collection), the **form** each contributor fills in, and the **approval** step that gates the write-back. Reporting sends information out; contribution brings it back in, through the same security and the same audit trail. See [Contribution overview](#).
- **AI context.** Reportworq's AI answers from the reports you have already built and approved, rather than querying your systems directly. That keeps answers traceable to something a person signed off, and keeps a user's AI access identical

to their normal access. An author steers it per report with an **AI profile**, which tells the AI what the report means and how to talk about it. This is the steering wheel that stops answers drifting. See [Create an AI Profile for a report and AI in Reportworq](#).

The value loop

These pieces serve one repeating cycle: an author **defines** a report once, Reportworq **generates** it (on demand or on a schedule) by refreshing the data and rendering each output, and then **distributes** the results to where people work. Because the report is defined once and reused, the authoring effort carries over to every future run. The full arc, from authoring through monitoring, is walked through in [The report lifecycle](#).

Where to go next

- See the arc in action in [The report lifecycle](#).
- Build your first report in [Your first report, end to end](#).
- Look up any term in the [Glossary](#).

Your first report, end to end

tutorial

report author

This tutorial walks the shortest complete path in Reportworq: create a report, point it at a source workbook, choose the worksheets to output, and preview a finished file. The worked example is a **monthly P&L pack** built from a template workbook. By the end you will have a saved report and a downloadable preview file.

Keep the mental model from [Key concepts](#) in mind: a report is a *template that makes outputs*. Here you build the template and preview one output. Setting output formats, destinations, and schedules comes later and is linked at the end, so nothing in this tutorial is sent to a recipient.

Prerequisites

- An account that can author reports, that is, an **Author** or an **Administrator**. Those accounts see the **New** control and the Job Editor; a **User** account does not.
- A report provider with at least one source file. A **report provider** is a configured location that holds source workbooks and templates. **A new server ships with no sample content**, so you will be pointing at your own source workbook. If no provider is configured, an administrator can set one up (see [The integrations hub](#) and [Connect a data source](#)).
- About 10 minutes.

Create the report

1. On the **Workspace**, open the **New** control and select **New Report**.
2. Enter a name, for example `Monthly P&L pack`, and select **OK**.

The report appears in the Workspace list with 0 components.

3. On the new report's row, select the **edit** action.

The **Job Editor** opens. It has a six-step rail, **Reports**, **Parameters**, **Bursting**, **Distribution**, **AI Profile**, and **Test**, and starts on the **Reports** step. Because the report is empty, the step shows *0 reports, 0 sheets*.

You will use only the **Reports** and **Test** steps in this tutorial.

Add a source workbook

The Reports step is where a report is bound to the content it renders. Its own description states the model: *"Source files and the worksheets they output, in order."*

1. On the **Reports** step, select **Add a Report**.

The **Select a Report** dialog opens and browses the report provider as a folder tree with a file grid.

2. In the provider tree, open your report provider, then select the source workbook in the file grid.
3. Select **OK**.

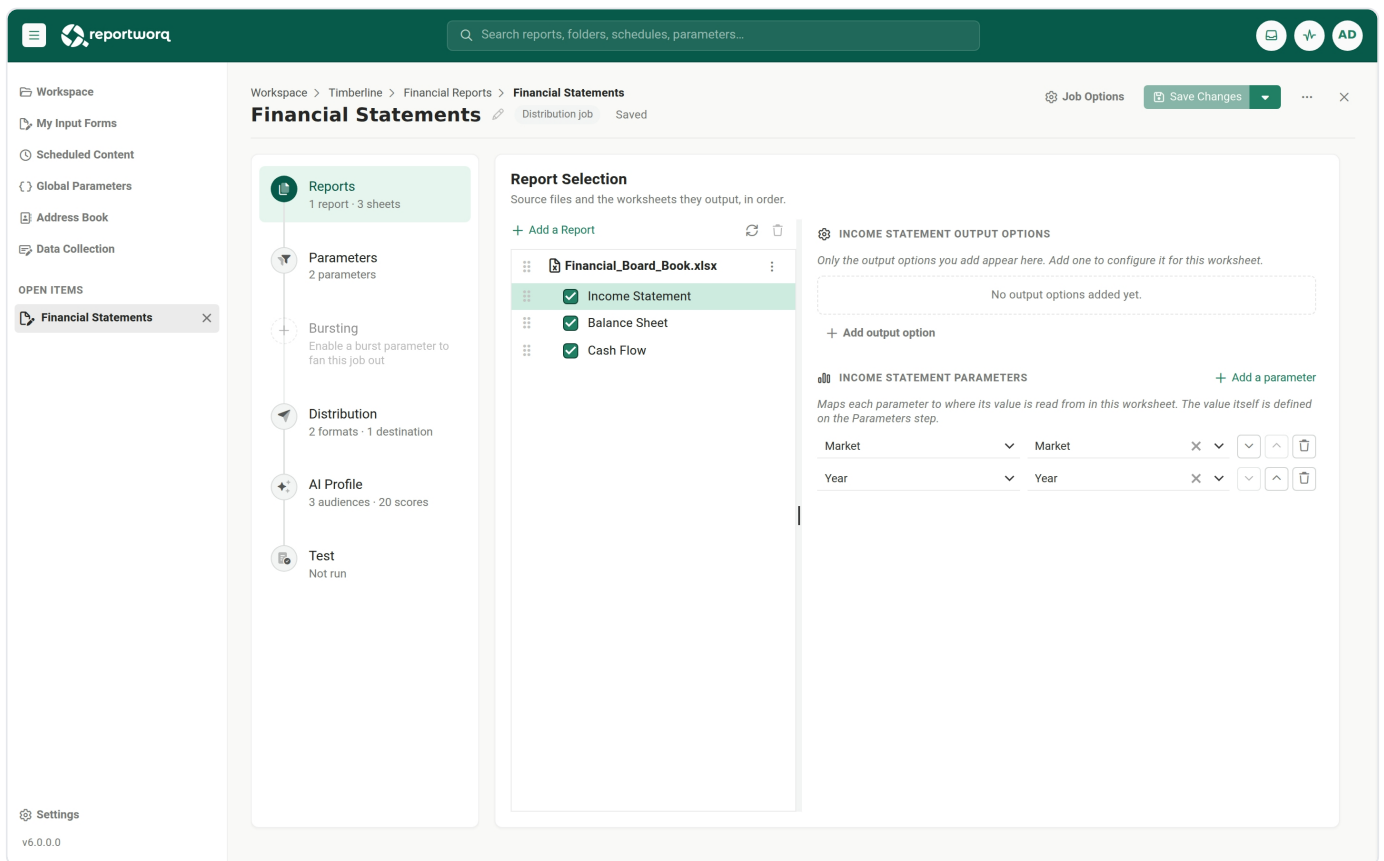
The workbook is added to the report, and its worksheets appear as a checklist on the Reports step.

Choose the worksheets to output

Each worksheet in the source workbook has an include checkbox. Included worksheets are rendered in the order they appear in the list.

1. Select the checkbox for each worksheet you want in the output. For a P&L pack, include the statement worksheet and its supporting tabs (for example, **P&L Template**, **Data**, and **Lookups**).
2. Clear the checkbox for any worksheet you want to leave out.

The step header updates to reflect the included worksheets, for example *1 report, 3 sheets*.



An excluded worksheet is dropped from the delivered file but **still calculates** during the run, so a hidden calculation or lookup tab can feed the visible sheets without appearing in the output. That is a deliberate technique, not a bug.

Preview the output

The **Test** step generates a preview through the same pipeline a production run uses, but with delivery suppressed. No file is emailed or written to a destination, so it is a safe way to confirm the report renders and the data resolves.

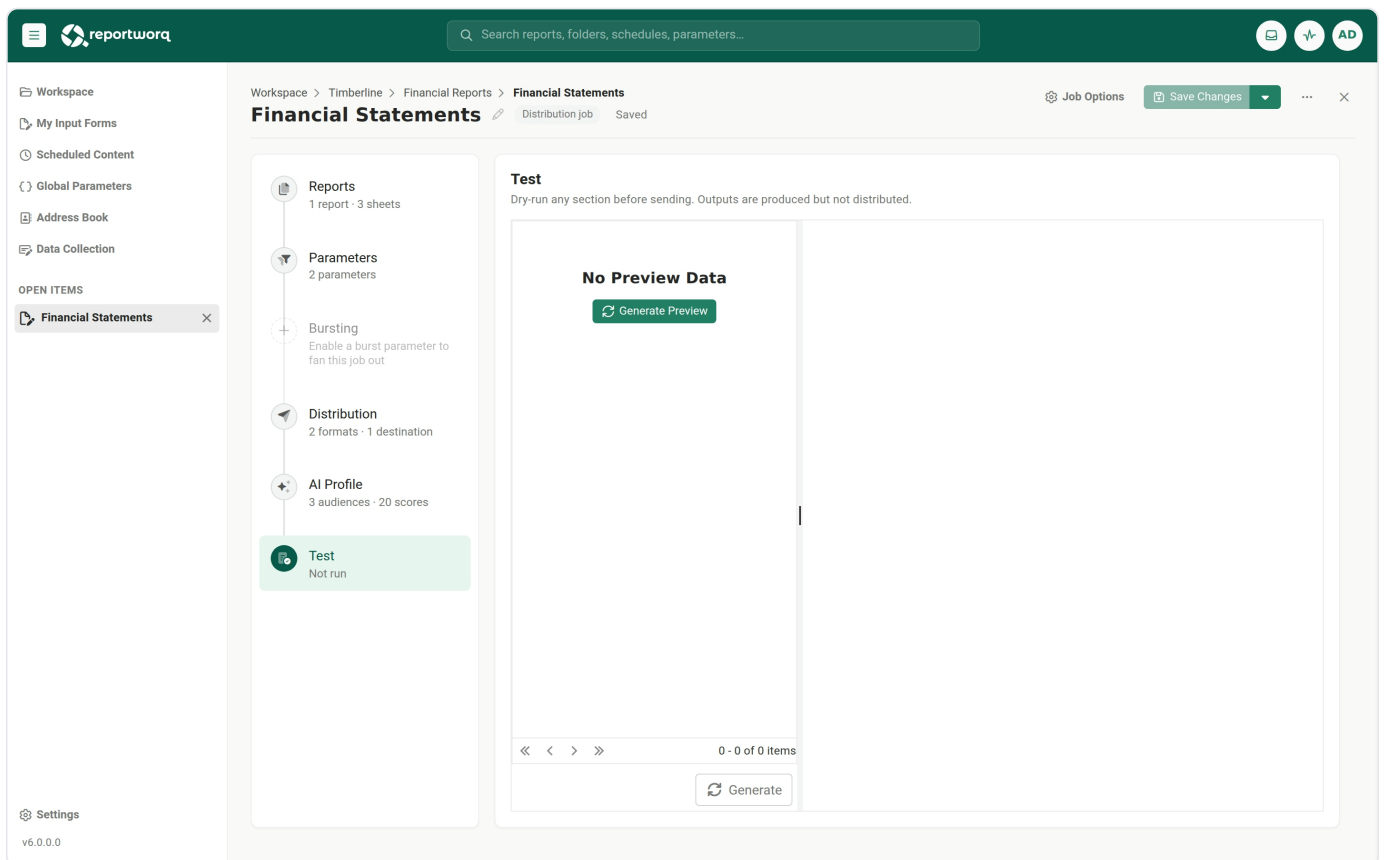
1. Go to the **Test** step.
2. Select **Generate**.

Reportworq builds one or more preview rows from the current configuration. The preview pane is a table of contents of the pages the report will produce, useful for checking sheet names and order before you run.

3. On a preview row, select **Run Preview Job**.

The row produces output. When the run finishes, it offers **Download job file** and **Download log report**.

4. Select **Download job file** to open your finished output, and **Download log report** to confirm the outcome. A successful run reports a datapoint summary such as *Success - 1 datapoints*, confirming the data resolved.



Save the report

- In the editor chrome at the top, select **Save**.

The report is saved and reusable. (**Save** is enabled only while the report has unsaved changes; adding the source workbook made it so.)

What to do next

You now have a saved report and a preview file. From here, extend the *same* report:

- Drive it by period, entity, or store with [Parameterize a report](#).
- Produce one personalized output per recipient with [What bursting is and when you need it](#).
- Set output formats and delivery targets with [Destinations](#).
- Run it on a recurring cadence with [Schedule a job](#).

For a narrative view of that whole arc, see [The report lifecycle](#).

Roles and what you can do

reference

report author

user

administrator

Reportworq tailors the app to each account. What a person sees, the side rail, the toolbars, the actions on a row, is decided by their **role**. Two things combine to produce what an account can actually do:

- **Role** decides the *shell* and the set of actions available (for example, whether the **New** button appears at all).
- **Permissions (ACLs)** decide *which items* the account can see and act on inside that shell.

An account's effective access is the role constrained by the permissions on each item. This page is a reference to the roles themselves.

The four roles

Reportworq has **three licensed roles**, Administrator, Author, and User, and one role that is not a license at all: the **Recipient**, who simply receives distributed content. Every person who signs in holds exactly one licensed role, and that role applies across the whole installation (you are not an Author in one workspace and a User in another).

Role	Licensed	Lands in	Can	Cannot
Administrator	Yes (seat)	The full shell plus Settings	Everything: manage security, users, integrations, connections, content, and licensing; reaches every workspace	(System-wide reach depends on administrator tier, see below)
Author	Yes (seat)	The full Workspace shell	Create, edit, run, schedule, and monitor reports, campaigns, and data models where their permissions allow; use the Excel authoring add-in; use global search; see (read-only) who has access to an item	Reach system settings unless also an administrator; see content outside the workspaces and folders granted to them; change anyone's permissions
User	Yes (seat)	A read-only Workspace, with My Input Forms added to the side rail when they are assigned to a contribution campaign	View permitted folders, reports, and outputs by ACL; fill in and submit contribution forms; refresh distributed PowerPoint decks in the add-in; reach content through the MCP server and the built-in AI assistant	Create or edit reports; the New button, authoring toolbar, and header search box are absent
Recipient	No (not an entitlement)	Nothing, they never sign in	Receive distributed output wherever they already work, email, Slack, Teams, SharePoint, a network folder, and more	Sign in to Reportworq, or see anything beyond what is delivered to them

What each role sees

- **Administrator.** An administrator gets the author experience plus the **Settings** surfaces (security, integrations, configuration), is the only role that can see any

content regardless of item permissions, and is the only role that can **change** who can see content.

- **Author.** The Workspace browser, the Job Editor, Scheduled Content, monitoring, the Address Book, and Global Parameters. This is the primary content-building role. An Author respects item permissions and workspace scope; it is not an administrator and does not bypass content security. An Author can **see** who has access to an item, read-only, but cannot change it.
- **User.** A User signs in to consume Reportworq rather than build it. Depending on what they are granted, a User browses a read-only Workspace to find reports and view their outputs, participates in contribution campaigns as a contributor or approver, refreshes a distributed PowerPoint deck through the add-in, or reaches governed content through the MCP server and the built-in AI assistant. In the read-only Workspace the row ⋮ menu is trimmed to **Properties** only (every other action is a change), and the shell omits the header search box and the Activity area. A User assigned to a contribution campaign keeps the same read-only Workspace and gains a **My Input Forms** entry in the side rail, where they fill in and submit the forms the campaign assigns. There is no separate contribution-only shell and no contribution-only account.
- **Recipient.** A recipient is not a Reportworq account and never signs in. They simply receive distributed output where they already work. Recipients are managed as contacts in the Address Book, they hold no role, consume no license, and are unlimited and free.

Recipient is a concept, not a setting. "Recipient" describes anyone who receives distributed content without a login. There is no Recipient entitlement to grant and no Recipient seat to buy; you never assign it in Security. Sending an output to someone does not require them to have a Reportworq role. See [Address book and contacts](#).

Who can see and change permissions

An item's permissions, the list of accounts and groups that can see a folder, a report, or the workspace as a whole, are themselves governed by role.

Role	Can see permissions	Can change permissions
User	No, anywhere. Properties shows the Overview tab only, with no Security tab.	No
Author	Yes, read-only, everywhere they are shown.	No
Administrator	Yes	Yes

As an Author you can open the **Security** tab of any item you can reach and see exactly who has access, which is often what you need in order to answer "why can't they see my report". The list is read-only: there is no **Save**, no way to add an account or group, and no way to remove one. Changing access is an administrator task, so ask an administrator.

This holds however you reach a security surface, including from a saved link or browser history.

Earlier role names fold in

If you are reading older documentation, license notes, or an account export, several retired names map onto the three current roles:

- **Reader, Consumer, Contribution End User**, and **PowerPoint End User** all fold into **User**. Contribution participation and PowerPoint add-in refresh are capabilities of the User role, not separate seats.
- **Reporting End User** and the "**Reporting add-in for Excel**" seat fold into **Author**. The Excel authoring add-in is part of the Author role.
- The **Author** entitlement was labeled **Reportworq Administrator** in earlier versions; it is the same entitlement under a clearer name. Where you see "Reportworq Administrator" used as a *user role*, read it as **Author**. It is not an administrator: it grants authoring, not administration, and it cannot change permissions.

Administration and workspace access

Administration is not tiered. The **Administrator** role has full access to every workspace on the server regardless of individual grants, and it governs roles and cross-workspace scope. There is no per-workspace administrator: Settings, including datasource configuration, is Administrator-only.

Within a workspace there is one kind of grant: everyone listed on a workspace is a **member**, and a member can reach the workspace and its content as their item

permissions allow. There is no second level to choose.

Workspace membership decides *whether* an account can enter a workspace; per-item permissions decide *which* items inside it the account can see and act on.

If you are looking for Configuration Access. Earlier versions offered a second workspace level, **Configuration Access** (sometimes called Workspace Administrator), that allowed datasource management inside one workspace. It has been removed. Member is the only level.

Notes and limits

- Roles constrain *access*, not *distribution recipients*. Sending an output to someone does not require them to have a Reportworq role; recipients are managed through contacts and the Address Book, and they consume no license.
- Contribution participation (contributor or approver) is a capability of the **User** role, available when the **Contribution** capability is licensed. It is not a separate seat.
- At least one **Administrator** must always exist; an account with no workspace access cannot sign in.
- How roles are assigned (directly, through groups, or through identity-provider claims) and how permissions are configured are administrator tasks. See [Sign-in and SSO](#) and [Workspaces](#).

Global parameters

[how-to](#)[report author](#)

A **global parameter** is a named parameter value defined once at the workspace level and reused across every report in that workspace. Instead of setting the same reporting period, version, currency, or entity on job after job, you define it once and bind reports to it. Rolling it forward is then a single edit, and Reportworq shows you exactly which reports the change affects.

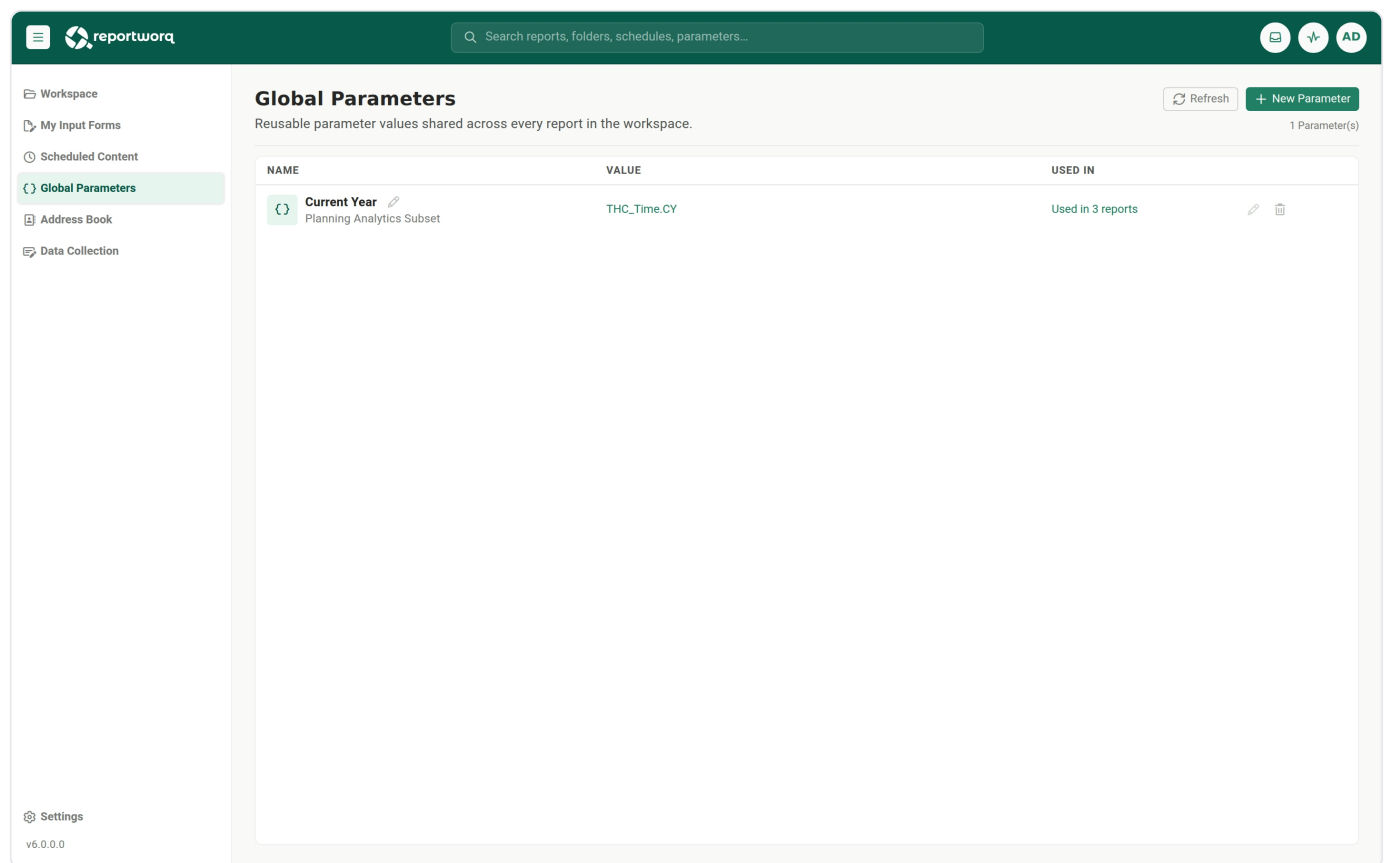
When to use it. Whenever the same parameter value is, or should be, identical across many reports: a **Current Fiscal Year** shared by every monthly pack, a **Reporting Version** (Actual, Budget, or Forecast) shared across a Planning Analytics suite, a common **Reporting Currency** or **Consolidation Entity**. For a value unique to one report, a local parameter on that report is simpler.

Open the screen from **Global Parameters** in the left side rail, grouped with the other admin items (Scheduled Content, Address Book, Data Collection). It is not shown in the read-only shell a **User** sees. The list has four columns: **Name**, **Value**, **Used In**, and row actions.

Before you begin

- You need administrator access to the workspace.
- A connector or plugin that supplies the parameter value type you want must be installed. The value types you can choose (for example a TM1, SQL, or text value) come from the connectors and plugins installed in your environment.

Create a global parameter



1. Select **New Parameter**.
2. Enter a unique **name**. Duplicate names are rejected.
3. The parameter is created with no value yet, its subtitle reads "No value set."
4. Select the value link (or the pencil action) to open the parameter sidebar, and set the value. It can be any parameter type your installed plugins provide, for example a TM1, SQL, or text value.

Reference it from a report

An author binds a report's parameter to the global value on the report's Parameters step. Once a report references the global parameter, it counts toward the **Used In** total. In the schedule editor, a global-backed parameter is marked with a **global** chip.

See where a parameter is used

The **Used In** column shows a live count of the reports that reference each parameter ("Used in N reports"), computed by scanning every job for the parameter name. Select the count to open a list of the referencing jobs with their names and paths. A parameter with no references shows "No references."

Use this before you touch a shared value, so you know its blast radius up front.

Change a value safely with impact review

Whenever a parameter is referenced by reports, renaming it, editing its value, or deleting it first shows an **Impact Review Required** confirmation that enumerates the affected reports and warns that the change affects future runs.

- **Rename** the parameter inline (click the name to edit). The rename is checked for duplicates, then the impact review lists the affected reports.
- **Edit the value** through the parameter sidebar. The same impact review appears when the parameter is referenced.
- **Delete** the parameter with a confirmation, preceded by the impact review if it is referenced.

The list refreshes automatically as entries change, and a manual **Refresh** action is available. Global parameters also appear in global search, so you can find one from the header search box.

Notes and limits

- Global parameters are **workspace-scoped**. Their values are not shared across workspaces.
- Changing a referenced value affects **all future runs** of every referencing report, the impact review exists precisely because the change is broad. Review the affected list before you confirm.
- Values can be read through the content-store API but are not API-writable, mutations go through the product only.

See it in action. [How Timberline rolls the reporting year forward](#) walks through the demo system's `Current Year` parameter, the four reports and campaigns that follow it, and a one-value change you can make to watch the numbers, the file names and the destination folders all move together.

Going deeper. For how an author binds a report parameter to a global value, and the full parameter model, see [Parameterize a report](#).

Replicate output from your source system

how-to

report author

Replication is one report producing many outputs from a parameter's own values. Set the parameter's type so it reads the values you want, set its **replication mode**, and Reportworq produces one output per value at run time. Because the values are resolved on every run, the list of outputs tracks your source system with nothing to maintain in Reportworq.

Replication does not use the **Bursting** step, and it does not need the **Burstable** flag. Leave that flag off. It exists only to unlock the Bursting step, which is for **burst sets**, the tool for controlling individual outputs.

Before you start

- The report is parameterized on the value you want to fan out on, for example **Cost Center**, **Entity**, or **Region**. See [Parameterize a report](#).
- The parameter's type can hold more than one value. Any of the list or query types works, including a Planning Analytics subset, dimension, or MDX query, a SQL query, a Workday Adaptive, Oracle Smart View, Anaplan, or Vena dimension, an Excel range, or a typed Text List. Only **Text Value** is fixed to a single value. For the full catalog, see [Parameters reference](#).
- A connection of the matching kind exists for any connector-specific type. A type appears in the list only when a connection of that kind is configured.
- The parameter is mapped on every worksheet that must replicate with it. See [Parameterize a report](#).

Set a parameter to replicate

1. Open the job and go to the **Parameters** step.
2. Find the parameter you want to fan out on and open its **Edit** sidebar.
3. Set **Parameter Type** to the type that reads your values, then fill in the type's own editor below it: pick a subset from the tree, write the MDX or SQL query, choose the dimension members, point at the Excel range, or enter the list one value per line.
4. Set **Replication mode**:
 - **One report per item** produces a separate output file per value. This is the choice for per-recipient delivery.
 - **One page per item** produces extra worksheets inside one output instead of separate files.
 - **No replication** declares the parameter a single value.
5. Leave **Enable this parameter to be used in burst sets** turned off.

6. Select **Save & Close**.

Save matters. The Edit Parameters sidebar commits only on **Save & Close**. If you close it with Escape or the X, your changes are discarded.

Where the values come from

The values are whatever the parameter's type resolves to at run time, so the choice of type is the choice of source. A Planning Analytics subset or MDX query reads the planning model live, so adding a member in Planning Analytics changes the roster with no change to the job. A SQL query reads the database on every run, so filtering to only the entities with activity this period makes the run scale up and down on its own. A Workday Adaptive, Oracle Smart View, Anaplan, or Vena dimension behaves the same way against those systems. An Excel range reads a workbook. A Text List is a fixed list you type.

The pattern is identical for all of them: whatever the type returns becomes the list of outputs. Choose the type that matches wherever your system of record already holds that list.

Route each output to a recipient

Recipients are optional. If the job delivers to a fixed destination, such as one network folder or a static email list, there is nothing to configure here.

When each output needs its own recipient, the usual approach for a replicated job is **contact mapping** on the Distribution step: it matches a value from the replicated parameter against a field on your contacts and uses the match as that output's recipient. See [Assign recipients to bursts](#).

What happens at run time

Reportworq resolves every **One report per item** parameter to its list of values and produces one output file per combination of those values. Each output gets a stable internal name that keys its own output folder and resolves the file-name tokens for that value, so a name built from the parameter produces a distinctly named file per output. See [Name and organize output files](#).

A **One page per item** parameter does not add files. It multiplies worksheets inside a single output.

The **Run Jobs** and **Activity** surfaces show a **Burst** chip on a job only when that job resolves to more than one output, so the chip reflects the actual fan-out for the run rather than any flag on the job.

Use cases

- **Per-cost-center P&L.** A **Cost Center** parameter typed as a Planning Analytics subset, set to **One report per item**, produces one P&L per cost center, and picks up new cost centers as the model gains them.
- **Per-entity forecast package.** An **Entity** parameter driven by a SQL query against the entity master produces one packet per active legal entity, with no list kept in Reportworq.
- **Per-region pack from a planning system.** A **Region** parameter typed as a Workday Adaptive dimension produces one pack per region member.
- **Monthly pack with two axes.** A **Period** Text List and an **Entity** subset, both set to **One report per item**, produce one file per period and entity pairing.
- **One workbook, a tab per department.** A **Department** parameter set to **One page per item** produces a single workbook with one worksheet per department instead of many files.

Notes and limits

- **Two or more replicating parameters cross-join.** Three business units and four departments produce 12 outputs. Check the arithmetic before you run.
- **A "One report per item" parameter that resolves to zero values fails the run**, with an error naming the parameter, rather than succeeding with nothing to deliver. A Text List whose lines are all blank resolves to zero values.
- **Map the parameter to every worksheet that must replicate with it.** Mapping it on one worksheet and not another leaves the second worksheet unreplicated. See [Parameterize a report](#).
- **A secured parameter is available only for "One report per item."** Every value in a one-page-per-item output stays in the same workbook, so there is no per-value file to gate.
- **If individual outputs need different treatment**, for example a different query or a hand-picked value for one of them, replication is the wrong tool. Build [burst sets](#) instead.

Going deeper. Embedding one parameter's value inside another parameter's query, with the `%param:%` and `%global:%` tokens, drives cascading one-report-per-item fan-out. See [Advanced parameters](#).

Address book and contacts

[how-to](#)[report author](#)

The **Address Book** is the workspace directory of **contacts**, the people distribution delivers to and contribution notifies. Each contact holds a name, a set of delivery destinations (email, Slack, Teams, SharePoint, network folder, SFTP, and so on), and optional custom metadata. Maintaining recipients here once, rather than per job, is what lets bursting and distribution map outputs to the right people automatically.

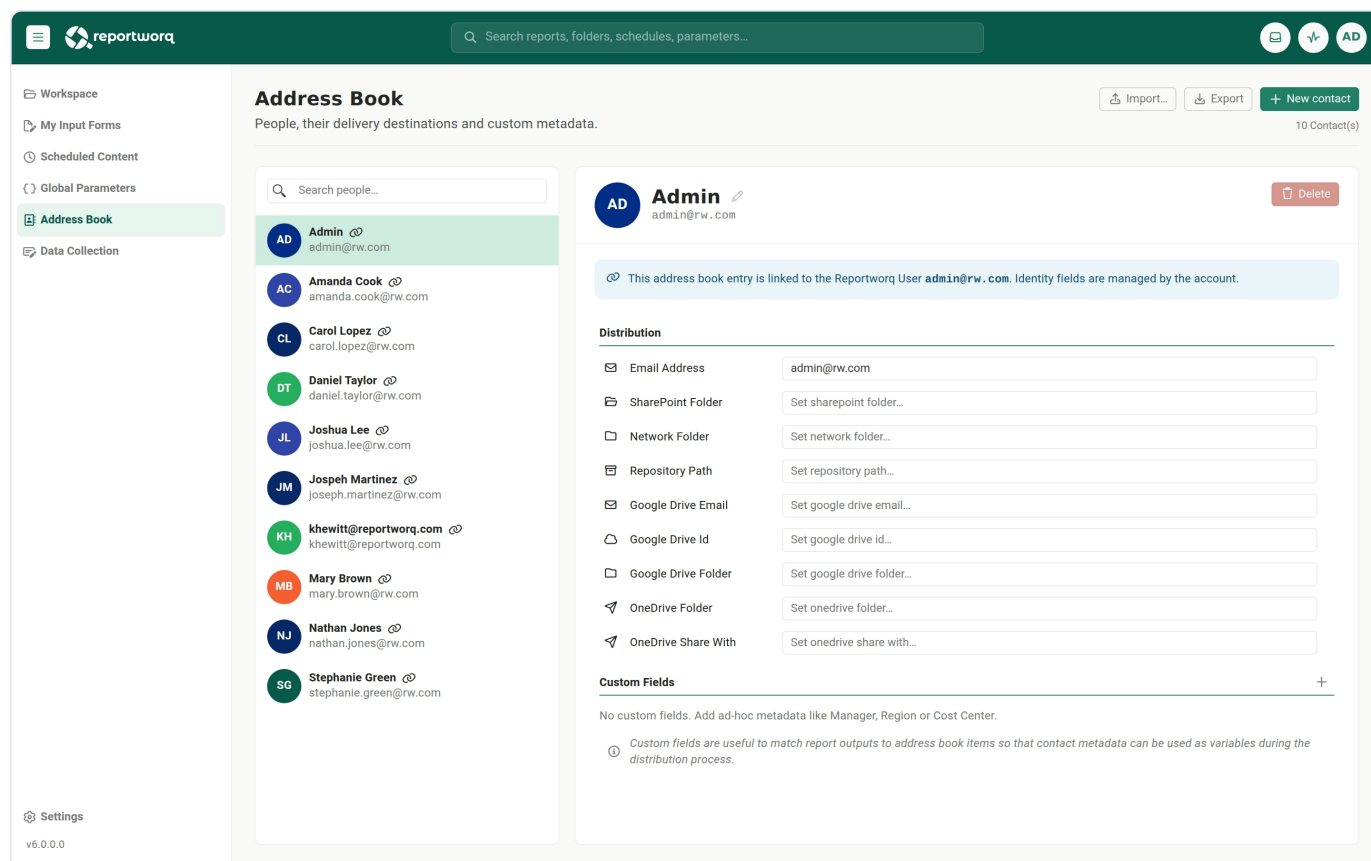
Open the screen from **Address Book** in the left side rail, grouped with the other admin items. It is not shown in the read-only shell a **User** sees. The layout is master-detail: a searchable contact list on the left, a person editor on the right. Contacts also appear in global search.

When to use it. Populate the Address Book before or alongside building distribution, so bursting can resolve each output to a contact's destinations, see [Assign recipients to bursts](#).

Before you begin

- You need administrator access to the workspace.
- The **distributors you enable determine which destination fields appear** on a contact. A distributor's field shows up only once that distributor is configured, see [The integrations hub](#).

Create and edit a contact



1. Select **New contact** and enter a name. Duplicate names are rejected.
2. Edit the name inline in the person editor. Fields are grouped into categories:
 - **Distribution**, one input per enabled distributor field, rendered in a canonical order (Email first, then Slack, Teams, SharePoint, Network, SFTP, Repository, Google Drive, then any others). Each value is free-form text but **must be a value the distributor accepts, or the job run errors**. Examples: a network folder as `\\MyServer\\MyFolder`, a SharePoint folder as a full `.../sites/...` URL, a Slack channel as `general`.
 - **Custom Fields**, ad-hoc per-contact metadata such as Manager, Region, or Cost Center. These match report outputs to contacts and can feed distribution variables.
 - **Access Rules**, shown only when a Workday Adaptive connection exists, per-dimension access rules resolved against Adaptive metadata.
3. **Delete** removes a non-linked contact after a confirmation.

A custom field only saves once a contact has a value in it

Adding a custom field creates a column, but the column is **dropped on save unless at least one contact has a value entered in it**. Adding the field alone does nothing, so enter a value for at least one contact immediately after you create the field, or it disappears.

Contacts linked to user accounts

A contact can be linked to a Reportworq user account. **Enabled accounts that have authentication credentials are listed automatically** as contacts, you do not add them by hand. A linked contact shows a link glyph and a banner noting that its identity fields are managed by the account. Linked contacts **cannot be deleted** here, the delete action is disabled with an explanatory tooltip.

Directly-created contacts get no entitlements

A contact you create directly in the Address Book (not through the authentication interface) can only be a **report-distribution recipient**. It acquires **no Reportworq entitlements or licenses**. Licenses and entitlements attach only to accounts added under authentication. The Address Book is a delivery directory, not an access-grant surface.

Import and export

Use **Export** to download all contacts as `contacts.xlsx`, and **Import** to ingest contacts from an uploaded `.xlsx` workbook (the first worksheet). The round-trip covers the name, tags, distributor destination values, and custom fields.

The import obeys a strict column contract:

Column prefix	Meaning	Behavior
<code>Id</code> , <code>Name</code>	Required identity columns	Import fails if either is missing. Rows match an existing contact by <code>Id</code> , then by <code>Name</code> . A blank or unmatched <code>Id</code> creates a new contact.
<code>D_...</code>	A built-in distributor field, formatted <code>D_<DistributorName>_<FieldId></code>	Sets that field only if the distributor and field both exist. A mistyped or unknown distributor or field is silently skipped , no error.
<code>P_...</code>	A custom field	Creates the custom field if it does not already exist, then sets its value.
<code>Tags</code>	Comma-delimited tags	Round-trips the contact's tags.

Import is additive and never deletes. It only creates or updates the rows present in the file, contacts absent from the file are left untouched. To remove a contact, delete it in the screen, do not omit it from an import expecting it to disappear.

Note: because a `P_` column creates the field, and a custom field is not saved until a contact has a value in it, an import that both adds a `P_` column and gives at least one row a value is the cleanest way to introduce a new custom field in bulk. Import to Excel and back is also the sanctioned workaround for bulk grouping, since contacts cannot be organized into folders in the screen.

How contacts get used downstream

- **Distribution contact mapping** resolves a burst output to a contact by matching a field to a value, and the field lookup is **case-insensitive**, so `NORTH`, `North`, and `north` all resolve to the same contact. See [Assign recipients to bursts](#).
- **Contribution notifications** resolve contributor and approver recipients from these contacts.

Notes and limits

- Contacts are **workspace-scoped**.
- Linked (account) contacts cannot be deleted here, and their identity fields are managed by the account.
- Contacts **cannot be organized into folders or groups** in the screen. Tags are the nearest classifier, set through the import and export `Tags` column.
- A distributor's destination field appears only after that distributor is enabled.

Going deeper. For how a bursted job routes each output to the right contact, including the burst-set-wins precedence over mapping, see [Assign recipients to bursts](#).

Excel: build and refresh reports

[how-to](#)[report author](#)

The Reportworq add-in puts a task pane on the right of the Excel grid, the **companion pane**. This page covers the reporting side of it: getting governed data onto a worksheet and keeping it current.

When to use it. Reach for the add-in when you want source-system data in a layout you control and refresh on demand, rather than authoring a full template-based job on the web.

Before you start

- The shared prerequisites in [The Office add-ins](#).
- **A data model with at least one published view**, or a direct-query-enabled model, that you are permitted to see.

Reports are built against a data model, not a raw connection

This surprises people, so it is worth stating plainly: **the add-in cannot connect to a raw datasource**. There is no "pick a database and write SQL" path in the pane. Everything you report on in Excel comes from a **data model**.

A data model is a governed, curated layer an administrator or author builds over a connection: it selects the tables or cubes worth exposing, renames technical columns into business field names, applies filters, and publishes one or more **views** over the result. A view is a named, ready-to-use field set.

That indirection is the point:

- **Field names make sense.** A report author picks *Cost Center* and *Actual Amount*, not `dim_cc.cc_code` and `f_gl.amt_base`.
- **The curation is done once.** Everyone reporting against that model gets the same definitions, so two reports cannot quietly disagree about what a field means.
- **Permissions are governed centrally.** You only see the models and views you are allowed to see, and the model owner controls what is exposed at all.
- **The underlying source can change** without every workbook breaking, because the model absorbs it.

So the first question when someone cannot find their data in the add-in is not "is the database connected" but "is there a published model view for it, and can this user see

it". To create or publish one, see [Data models](#).

Find your way around the pane

ON THIS WORKBOOK



New report

Connect to data and import a list



New chart

Build a chart or map from a range and import it



Nothing on this workbook yet

Start with New report or New chart to add content to this workbook.



Formula library

20 capabilities · Reporting · Contribution · Data



- **Home** lists the reports stored on the active workbook, grouped by worksheet, each with a refresh control. Per-report actions are Edit, Refresh, Clear Data, Move Report, and Delete, plus workbook-level **Refresh all data** and **Clear all data**.
- **New report** launches the import stepper.
- **New chart** launches the chart and map builder, see [Excel: charts and maps](#).
- The **Formula Library** dock is pinned at the foot of the pane, see [Excel: find and insert functions](#).

Editing a report **saves continuously** as you move between screens; there is no Save button. Refresh is the one explicit action that writes data into the worksheet.

Import a published data-model view

1. On **Home**, choose **New report** to open the **Connect > Fields > Place** stepper.
2. **Connect**. Browse **Workspace > Folder > Data Model > View** and pick a **view** (a predefined field set) or a **model** (which adds a **Query** step for an ad-hoc query). Only models and views you are permitted to see appear.

3. **Fields.** Select which fields become columns. Switch to **Arrange** to reorder, rename, hide, and configure them.
4. **Place.** Set the target cell or range (type an address or use **Select Range**), and optionally import as an Excel Table, choose an **Insert Behavior**, auto-resize columns, or update a named range.
5. Choose **Import to {cell}**. Reportworq runs the model, writes the result, returns you to Home, and adds the report to the list.

Three placement rules are worth knowing before they bite:

- A named range you point at with **Update named range must already exist**. Reportworq does not create it.
- That named-range update takes effect when the report runs **as a job**, not on an in-Excel refresh.
- **Max Preview Rows** caps only the in-Excel preview. It does not limit what the job renders.

Shape the list report

Within the **Fields** step, open any column to configure it in the field editor.

Control	What it does
Display mode	Normal , Group (collapse repeated values down the column), or Section (a section break with its own sorting and layout). Only one field may be a Section at a time.
Sort	A sort mode, plus a priority when more than one field sorts.
Aggregate	Sum and Average on numeric columns; Min and Max on numeric or date columns; Count and Count Distinct on any type.
Filter	An operator and a value: a numeric literal such as <code>>=5</code> , single-quoted text such as <code>'North'</code> , or a variable such as <code>{{Region}}</code> . Combine rows with AND/OR.
Formatting	Header, Data, and Subtotals, each with its own format mode.

Two habits pay off. A **hidden field** (the eye toggle) still filters and sorts without appearing in the output, which is how you filter on something the reader should not see. And **selecting only the fields you need** reduces the work the datasource does, which is the cheapest performance win available here.

Filters that follow a parameter

The `{{Variable}}` filter form is what connects an add-in report to the rest of Reportworq. Filter a column on `{{Region}}` , and when the report runs as part of a job the value comes from the job's parameter, so the same workbook produces a different

slice per region. See [Parameterize a report](#).

Refresh a report

- The **first** refresh of a report must be **from source**, which runs the query.
- **From cached** re-renders from the last query result. Fewer queries, faster, and the right choice while you are adjusting layout.
- **Clear all data** removes a report's column data but keeps its headers, which come back on the next refresh.

Notes and limits

- The pane is a narrow single-column surface, not a full-page editor. It does not replace the web Job Editor for bursting, distribution, or scheduling.
- A data model is required. There is no connect-to-raw-datasource path in the add-in.

Going deeper. An imported list report is one of two ways to put source data on a sheet. The other is a Reportworq worksheet function written straight into a cell. For how the two compare and the full function catalog, see [Functions overview](#). To find and insert a function without leaving Excel, see [Excel: find and insert functions](#).

General report and layout functions

reference report author

The general family is the `RW.Tools.*` functions that **post-process the generated workbook**. They do not fetch data; they reshape and format what the data functions produced. Reportworq runs them in a fixed order during report generation, then clears each formula cell on success.

Every function has a canonical dotted name and a legacy short alias. Both resolve to the same function, and new workbooks should use the canonical form.

Function	Alias	What it does
<code>RW.Tools.Sort</code>	<code>RWSORT</code>	Sorts a range or table, optionally keeping only the top or bottom N rows
<code>RW.Tools.Suppress</code>	<code>RWSUPPRESS</code> , <code>RWZERO</code>	Hides or deletes rows or columns on a condition
<code>RW.Tools.Group</code>	<code>RWGROUP</code>	Builds Excel outline groups from repeated identifiers
<code>RW.Tools.BlankRow</code>	<code>RWBLANKROW</code>	Inserts blank rows next to the formula row
<code>RW.Tools.PageBreak</code>	<code>RWPAGEBREAK</code>	Inserts a horizontal or vertical page break
<code>RW.Tools.Resize</code>	<code>RWRESIZE</code>	Auto-fits a row height or column width, including merged cells
<code>RW.Tools.RemoveFormula</code>	<code>RWREMOVEFORMULA</code>	Replaces formulas with their values
<code>RW.Tools.SheetName</code>	<code>RWSHEETNAME</code>	Sets the worksheet output name from a formula
<code>RW.Tools.SheetColor</code>	<code>RWSHEETCOLOR</code>	Sets the worksheet tab color from a formula
<code>RW.Tools.SheetSuppress</code>	<code>RWSHEETSUPPRESS</code>	Suppresses a worksheet from a formula

The layout pipeline order

When several of these functions interact, order matters. Reportworq runs them in this sequence on the expanded data:

1. Data importers (`RW.Relational.Export` , `RW.PlanningAnalytics.MDX`) populate the ranges.
2. `RW.Tools.Sort`
3. `RW.Tools.BlankRow`

4. `RW.Tools.Group`
5. `RW.Tools.Suppress`
6. `RW.Tools.PageBreak` and `RW.Tools.RemoveFormula`

This guarantees, for example, that a top-N sort is applied before rows are suppressed.

RW.Tools.Sort

Sorts a target range or named table, with an optional limit that keeps only the top or bottom N rows after sorting. Use it instead of a separate sort-then-suppress pair when you only need the ranked portion of a dataset.

Signature

```
=RW.Tools.Sort(sortRange, [key1], [ascending1], [key2], [ascending2], [key3], [ascending3], [limitType], [limitCount])
```

#	Argument	Type	Default	Meaning
1	<code>sortRange</code>	String or range	<i>required</i>	A cell address such as <code>"A1:D30"</code> , resolved on the sheet holding the formula, or a named table. A table sorts its data rows, with the header excluded.
2	<code>key1</code>	Integer	<code>0</code>	Primary sort column, 1-based and relative to the sort range . <code>0</code> skips this level.
3	<code>ascending1</code>	Boolean	<code>TRUE</code>	<code>FALSE</code> sorts descending.
4	<code>key2</code>	Integer	<code>0</code>	Secondary key, applied only when <code>key1</code> values tie.
5	<code>ascending2</code>	Boolean	<code>TRUE</code>	Direction for <code>key2</code> .
6	<code>key3</code>	Integer	<code>0</code>	Tertiary key, applied only when <code>key1</code> and <code>key2</code> both tie.
7	<code>ascending3</code>	Boolean	<code>TRUE</code>	Direction for <code>key3</code> .
8	<code>limitType</code>	String	<i>none</i>	<code>"TOP"</code> keeps the first N rows after sorting, <code>"BOTTOM"</code> the last N. Case-insensitive.
9	<code>limitCount</code>	Integer	<i>none</i>	How many rows to keep. Must be a positive whole number.

Column indexes are relative to the range. If `sortRange` starts at column D, then `key1 = 1` means column D and `key2 = 2` means column E.

The limit arguments are positional. `limitType` and `limitCount` must be supplied together as the **last two** arguments. Reportworq detects them automatically, so you do not need placeholder arguments, and you can apply a limit without sorting at all.

Rows outside the limit are hidden, not deleted. They behave like any other hidden Excel row: present in the workbook, not printed or displayed. If `limitCount` is greater than or equal to the number of rows in the range, nothing is hidden.

Examples

```
=RW.Tools.Sort("A1:D30")
=RW.Tools.Sort("A1:D30", 1, TRUE, 2, FALSE)
=RW.Tools.Sort("A1:D30", 1, TRUE, "TOP", 5)
=RW.Tools.Sort("A1:D30", 1, FALSE, "BOTTOM", 3)
=RW.Tools.Sort("A1:D30", "TOP", 5)
```

The fourth example sorts descending and then keeps the last three rows, which are the three smallest values. The fifth applies a limit to the data in whatever order it already has.

When an argument is wrong

Situation	Result
<code>sortRange</code> omitted	Formula error. It is required.
More than nine arguments	Formula error, too many parameters.
A key is <code>0</code>	That sort level is skipped.
<code>limitCount</code> is zero, negative, or fractional	The TOP/BOTTOM pattern is not recognized, the arguments fall through to sort-parameter parsing, and you get a parse error.
<code>limitCount</code> is at least the row count	Nothing is hidden.

RW.Tools.Suppress

Hides or deletes rows or columns based on a Boolean condition.

Signature

```
=RW.Tools.Suppress(doSuppress, [type], [mode])
```

#	Argument	Type	Default	Meaning
1	<code>doSuppress</code>	Boolean	<code>FALSE</code>	Suppression is applied when <code>TRUE</code> .
2	<code>type</code>	String or Boolean	<code>"rows"</code>	<code>"rows"</code> or <code>TRUE</code> for rows, <code>"columns"</code> or <code>FALSE</code> for columns.
3	<code>mode</code>	String or Boolean	<code>"delete"</code>	<code>"delete"</code> or <code>TRUE</code> removes; <code>"hide"</code> or <code>FALSE</code> hides.

The default mode is delete, not hide. This is the single most common surprise in the function surface. If you only want the rows out of sight, pass the hide mode explicitly.

Processing order protects coordinates. Rows are processed sheet by sheet from the bottom upward, and columns from the right toward the left, so earlier deletions never invalidate later coordinates. Adjacent formulas that share a sheet, direction, `doSuppress` value and mode are coalesced into one contiguous suppression range.

Because rows and columns can be deleted, later layout steps must not depend on the original coordinates.

Examples

```
=RW.Tools.Suppress(A1=0)
=RW.Tools.Suppress(TRUE, "rows", "delete")
=RW.Tools.Suppress(TRUE, "columns", "hide")
```

When an argument is wrong

Situation	Result
You expected hide and did not pass a mode	The rows or columns are deleted.
You expected top-down delete order	Rows go bottom-up, columns right-to-left.
You used hide but need them gone	Hidden rows still exist in the file. Use delete.

RW.Tools.Group

Creates Excel outline groups from repeated row or column identifiers.

Signature

```
=RW.Tools.Group(identifier, [axis], [visible], [additionalGroupDepth])
```

#	Argument	Type	Default	Meaning
1	identifier	String, number or Boolean	<i>required</i>	The group key for this row or column. Blank means do not group this position.
2	axis	String	"rows"	"rows" or "columns" .
3	visible	String	"show"	"show" starts the group expanded, "hide" starts it collapsed.
4	additionalGroupDepth	Integer	0	Adds extra outline depth over the same body range.

The first match is the header and stays outside the group. Consecutive matching non-blank identifiers form one group, and the first of them is treated as the header or summary line rather than being placed inside the collapsible body. This trips people up when they expect the whole block to collapse.

A blank identifier ends the current group without inserting a separator. A change straight from one non-blank identifier to another inserts a blank separator row or column so Excel keeps the two groups distinct.

Nesting. For row grouping, each helper **column** is a separate nesting level; for column grouping, each helper **row** is. Outer levels are processed before inner ones. Put the helper formulas in hidden helper columns or rows when you do not want them visible.

Examples

```
=RW.Tools.Group("East")
=RW.Tools.Group("Product A", "rows", "hide")
=RW.Tools.Group("Quarter", "columns", "show", 1)
```

When an argument is wrong

Situation	Result
Blank identifiers in the middle of a block	The group ends at the blank.
Expecting the first matching row to collapse	It is the summary line and stays outside the body.
Invalid <code>axis</code> or <code>visible</code>	Reportworq logs a warning and falls back to the default.

RW.Tools.BlankRow

Inserts one or more blank rows next to the row containing the formula. Rows are inserted after all runtime expansion and before suppression, so spacing works in a report whose size is only known at run time.

Signature

```
=RW.Tools.BlankRow([doInsert], [count], [mode], [insertAbove], [rowHeight])
```

#	Argument	Type	Default	Meaning
1	doInsert	Boolean	TRUE	FALSE makes the formula a no-op, which is how you drive insertion from another cell.
2	count	Integer	1	A whole number from 0 to 1000.
3	mode	String	"blank"	"blank" gives unformatted rows; "copyformat" inherits the style of the row immediately above the insertion point.
4	insertAbove	Boolean	TRUE	TRUE inserts above the anchor row and shifts it down, FALSE inserts below.
5	rowHeight	Number	none	Explicit height in points. Must be non-negative.

The **anchor row** is the row holding the formula. Insertions are processed top to bottom, and each one shifts the anchors below it, so row numbers stay correct without any manual adjustment.

Several BlankRow formulas on one row consolidate into one instruction. count becomes the maximum across them; mode, insertAbove and rowHeight are taken from the formula in the **leftmost** column; and doInsert is true if any of them is true.

Examples

```
=RW.Tools.BlankRow()  
=RW.Tools.BlankRow(TRUE, 3)  
=RW.Tools.BlankRow(TRUE, 2, "blank", FALSE)  
=RW.Tools.BlankRow(TRUE, 2, "copyformat")  
=RW.Tools.BlankRow(TRUE, 1, "blank", TRUE, 18)  
=RW.Tools.BlankRow(A1, 2)
```

When an argument is wrong. Invalid arguments never fail the report. Reportworq logs a warning and uses a safe default.

Situation	Result
<code>count</code> is fractional, for example <code>1.5</code>	Warning, treated as <code>0</code> , nothing inserted.
<code>count</code> is negative	Warning, treated as <code>0</code> , nothing inserted.
<code>count</code> is over 1000	Warning, capped to <code>1000</code> .
<code>mode</code> is neither accepted value	Warning, defaults to <code>"blank"</code> .
<code>rowHeight</code> is negative or non-numeric	Warning, height not set.

RW.Tools.PageBreak

Inserts a horizontal or vertical page break relative to the row or column holding the formula.

Signature

```
=RW.Tools.PageBreak(doBreak, [mode], [position])
```

#	Argument	Type	Default	Meaning
1	<code>doBreak</code>	Boolean	<i>required for any effect</i>	<code>TRUE</code> inserts the break.
2	<code>mode</code>	String or Boolean	<code>"horizontal"</code>	<code>"horizontal"</code> or <code>TRUE</code> , <code>"vertical"</code> or <code>FALSE</code> .
3	<code>position</code>	String or Boolean	<code>"before"</code>	<code>"before"</code> or <code>TRUE</code> , <code>"after"</code> or <code>FALSE</code> .

A horizontal break lands at the formula row, and an `after` break one row later. A vertical break lands at the formula column, and an `after` break one column later.

A break that conflicts with Fit To Pages is rejected. If the sheet's print setup is set to fit a number of pages tall or wide, the conflicting break direction does not take effect.

Passing `FALSE` does nothing. It does **not** remove a page break that is already in the template.

Place the formula after your dynamic data area so the break tracks the generated layout. The Boolean forms work, but the string forms read better in a template.

Examples

```
=RW.Tools.PageBreak(TRUE)
=RW.Tools.PageBreak(TRUE, "horizontal", "after")
=RW.Tools.PageBreak(A1, "vertical", "before")
```

RW.Tools.Resize

Auto-fits the row height or column width at the formula's location.

Signature

```
=RW.Tools.Resize([type], [pad])
```

#	Argument	Type	Default	Meaning
1	type	String or Boolean	"rows"	"rows" or TRUE resizes the row, "columns" or FALSE the column.
2	pad	Number	0	Extra size added after the auto-fit. Only numeric values have any effect.

Merged cells are auto-fitted. When the resize cell is part of a merged range, the function fits the whole merged range, and even a single unmerged cell has each line fitted correctly. WrapText must be on, and the merged autofit keys off the `RW.Tools.Resize` cell itself being part of the merged range. This is what makes it work for wrapped narrative, for example a TM1 commentary cube cell rendered to PDF. There is no longer any "cannot resize merged cells" limitation.

Hidden rows and hidden columns are skipped.

Examples

```
=RW.Tools.Resize()
=RW.Tools.Resize("columns")
=RW.Tools.Resize("rows", 5)
```

RW.Tools.RemoveFormula

Replaces formulas with their current values across a cell, range, row, column, or multi-range selection. Use it to freeze calculated results before distribution.

Signature

```
=RW.Tools.RemoveFormula(doRemove, targetRange, [mode])
```

#	Argument	Type	Default	Meaning
1	<code>doRemove</code>	Boolean	<i>required</i>	<code>TRUE</code> converts formulas to values.
2	<code>targetRange</code>	Cell, range, row, column or multi-range	<i>required</i>	The area to process.
3	<code>mode</code>	String or Boolean	<code>"all"</code>	<code>"all"</code> or <code>TRUE</code> removes every formula; <code>"provider"</code> or <code>FALSE</code> removes only TM1 provider formulas.

Entire-row references expand to every used column on the sheet, and entire-column references to every used row. Multi-range references such as `{A1:A2,B3:B4}` are supported. Non-formula cells are ignored.

Use `provider` mode when you want to strip provider formulas but keep the workbook's own Excel formulas working.

Examples

```
=RW.Tools.RemoveFormula(TRUE, A1)
=RW.Tools.RemoveFormula(TRUE, A:A)
=RW.Tools.RemoveFormula(TRUE, {A1:A2,B3:B4}, "provider")
```

When an argument is wrong

Situation	Result
<code>targetRange</code> omitted	Formula error. At least two arguments are required.
You expected blank cells to change	Non-formula cells are ignored.
You used <code>provider</code> mode on ordinary formulas	Those formulas are left in place.

The three sheet functions

`RW.Tools.SheetName`, `RW.Tools.SheetColor` and `RW.Tools.SheetSuppress` set a worksheet property from inside Excel, for cases where the value is only known at run time. All three share one rule set:

- **Only one of each is allowed per worksheet.** If a sheet carries two of the same function, **none** of them applies and every one is flagged as an error.

- **The Reportworq UI wins.** If the equivalent worksheet output option is configured in the job, the formula is ignored and flagged as an error.
- The three are evaluated independently, so an invalid `RW.Tools.SheetSuppress` does not stop the name or color from being applied.

RW.Tools.SheetName

```
=RW.Tools.SheetName(value)
```

`value` must evaluate to a valid worksheet name: not blank or whitespace, and still legal after Reportworq applies its normal sheet-name sanitization, which means it has to satisfy Excel's length limit and disallowed-character rules.

```
=RW.Tools.SheetName("P&L")  
=RW.Tools.SheetName(A1)  
=RW.Tools.SheetName("Sales - " & B1)
```

Error messages you may see:

```
RW.Tools.SheetName was ignored because Sheet Name is configured in the Reportworq worksheet options.  
Multiple RW.Tools.SheetName formulas were found on this worksheet. Only one is allowed, so no formula-based sheet name was applied.  
RW.Tools.SheetName did not return a valid worksheet name.
```

RW.Tools.SheetColor

```
=RW.Tools.SheetColor(value)
```

`value` can be an integer color value, a hex color such as `#FF0000`, or a named color such as `Red`.

```
=RW.Tools.SheetColor("#4F81BD")  
=RW.Tools.SheetColor(A1)  
=RW.Tools.SheetColor("Red")
```

A value that cannot be applied leaves the tab color unchanged and flags the formula, for example:

```
Error setting tab color on sheet 'Sheet1' using value 'NotAColor'.
```

RW.Tools.SheetSuppress

```
=RW.Tools.SheetSuppress(value)
```

TRUE or 1 suppresses the sheet. Any other value leaves it in place.

```
=RW.Tools.SheetSuppress(TRUE)
=RW.Tools.SheetSuppress(A1)
=RW.Tools.SheetSuppress(COUNTIF(A1:A10, "Error") > 0)
```

The third example is the useful pattern: drop a whole sheet from the output when the data behind it failed a check.

Import-by-name and copied references

These rules apply when a data importer targets a table or a copied range, and they are repeated here because they surface as layout problems:

- **A report table imported by name needs "Show Header Row" enabled.** The import writes into the table body below the header.
- **Use INDIRECT("...") for copied relative references** so a filled-down formula does not drift. The modern answer, though, is to **copy the sheets as a group**, which rewrites cross-sheet references correctly so you rarely need INDIRECT for that case.

Use cases

- **Cost-center report with outlines.** RW.Tools.Group outlines rows by department, collapsed by default, and RW.Tools.Resize auto-fits the wrapped narrative header.
- **Ranked P&L pack.** RW.Tools.Sort ranks cost centers descending and keeps the top 10, RW.Tools.Suppress deletes the zero rows, and RW.Tools.PageBreak starts supporting schedules on a new page.
- **Per-sheet formatting without the UI.** A template uses RW.Tools.SheetName("Sales - " & B1) and RW.Tools.SheetColor("#4F81BD") to name and color tabs dynamically.
- **Locked-down distribution.** RW.Tools.RemoveFormula freezes the calculated values before the workbook is emailed, so recipients cannot see or break the formulas.

Notes and limits

- The functions are inert in desktop Excel. They only run inside a Reportworq job.

- The sheet functions are ignored when the equivalent worksheet output option is set in the job, and only one of each kind is allowed per worksheet.
- Suppression defaults to delete. Be deliberate about hide versus delete.
- Version availability is approximate: the older layout functions date from around v5.0.0.60, `RW.Tools.RemoveFormula` from around v5.0.0.82, and `RW.Tools.Group`, `RW.Tools.BlankRow` and the sort limit arguments from around v5.0.0.91.

Related pages

- [Functions overview: which one do I use](#)
- [IBM Planning Analytics functions](#) and [Relational database functions](#) for the data importers these functions shape.
- [Worksheet output options](#) for the UI equivalents of the sheet-name, color, and suppression functions.

Schedule a job

[how-to](#)[report author](#)[administrator](#)

A schedule tells Reportworq **when** to run **which** jobs, with **what** parameter values and delivery behavior, so reports run unattended on a cadence instead of being kicked off by hand. You build a schedule in the **Schedule editor**, which defines one schedule as a single cron expression, a selection of jobs or folders, an optional set of parameter overrides, and delivery options.

When to use it. Once a job and its distribution are built and tested, schedule it for any report that has to run on a cadence: a nightly forecast refresh, a month-end P&L distribution, a weekly store pack. Scheduling is one of the most underused features in Reportworq. Many teams keep running jobs manually because the job "doesn't feel dynamic enough" to trust on a schedule, but a well-parameterized job runs the same way every time. Even scheduling to **Hold for Delivery** is worth it: it is the same as running a job and reviewing the result before it goes out, done automatically on your cadence.

Before you begin

- You need at least one runnable job. For output to be delivered, the job needs a configured distribution.
- Times in the editor are **server time** (the Reportworq host's time zone).

Open the editor and name the schedule

1. From **Scheduled Content**, select **New schedule**, or open an item's **Schedule** properties tab and choose **New schedule**. To change an existing schedule, select its row.
2. Enter a **name** in the header. The name commits when you press Enter or move focus off the field.
3. Use the **enabled** toggle to control whether this schedule runs. A disabled schedule is kept but does not fire.

Set the cadence with cron

Timing is a single **5-field cron expression** in the order `minute hour day-of-month month day-of-week`.

1. In **When it runs**, pick a builder preset: **Minutes**, **Hourly**, **Daily**, **Weekly**, **Monthly**, **Yearly**, or **Custom**. Each preset shows only the fields it needs and writes the cron for you.

2. Turn on **Multiple values** for a field to select more than one value (a comma list), for example several weekdays.
3. To type the expression directly, edit the raw cron string. Doing so switches the builder to **Custom**. For example, `0 2 * * *` runs every day at 02:00.
4. Check the **Next runs** preview and the human-readable description below the field. Both are in server time. A red field with "No upcoming runs, check the expression" means the cron is invalid and Save is disabled.

Reportworq uses standard 5-field cron. It does **not** use the old `? / L / W / #` tokens, month or day names, or seconds. The finest granularity is one minute. When both day-of-month and day-of-week are restricted, a day matches if **either** field matches.

The screenshot shows the 'Board Packet Delivery' configuration page in Reportworq. The page is divided into several sections:

- Workspace:** Includes links to Workspace, My Input Forms, Scheduled Content (3 items), Global Parameters, Address Book, and Data Collection.
- Scheduled Content:** Shows 'Board Packet Delivery' with a 'Disabled' toggle and a 'Save Changes' button.
- WHEN IT RUNS:**
 - Frequency: Weekly (selected), with options for Minutes, Hourly, Daily, Monthly, Yearly, and Custom.
 - Multiple values: A toggle switch is present.
 - ON DAYS: A row of buttons for Sun, Mon, Tue, Wed, Thu (selected), Fri, and Sat.
 - at: A time selection field showing 08:00.
 - Summary: 'At 08:00 AM, only on Thursday'.
 - Next 5 Runs (Server Time):

	Run Date	Time
NEXT	Thu Aug 6	08:00 AM
+1	Thu Aug 13	08:00 AM
+2	Thu Aug 20	08:00 AM
+3	Thu Aug 27	08:00 AM
+4	Thu Sep 3	08:00 AM
- ADVANCED OPTIONS:** A section with a 'No notifications' status.
- WHAT IT RUNS:** A searchable tree of folders and jobs. The tree shows:
 - Examples
 - Timberline
 - Financial Reports
 - Source Files
 - Board Packet (checked)
 - Board Packet with Division Slides
 - Commission Statements
 - Data Validation Check
 - Divisional Packet (checked)
 - Divisional Packet - Burst Set Example
 - Financial Statements
 - Financial Statements - Job Failure
- PARAMETER OVERRIDES:** A table with columns NAME, DESCRIPTION, and OVERRIDE VALUE.

NAME	DESCRIPTION	OVERRIDE VALUE
Region	Used by 2 jobs	not specified...
Year	Used by 1 job	not specified...
Market	Used by 1 job	not specified...

Choose what runs

1. In **What it runs**, use the searchable tree of folders and jobs.
2. Select individual jobs with their checkboxes. The summary reads like "N jobs · M dynamic folders."
3. To run every job in a folder as resolved at run time, select the folder as a **dynamic** query. It shows a **Dynamic** pill and a runtime-count hint, for example "~8 jobs today," so newly added jobs in that folder are picked up automatically.

Override parameters (optional)

1. In the **Parameter Overrides** inspector, review the parameters used across your selected jobs. A **global** chip marks a parameter backed by a [Global Parameter](#).
2. Enter a value to apply it to every selected job that uses that parameter. Leave it blank to keep each job's own default.

Use this to run several jobs for the same reporting period from one schedule: override the shared `Period` parameter once and every job runs for that period.

Set delivery behavior and save

1. In the advanced options under **When it runs**, set delivery and logging:
 - **Hold for Delivery** produces the outputs but holds them for release instead of distributing. A scheduled held run follows the same lifecycle as an ad-hoc hold: outputs are generated and logged, distribution is skipped, and the run lands in **Activity** under **Hold** to be released or canceled.
 - **Notifications** control when a run summary is sent.
 - **Verbose logging** and **Trace logging** capture extra diagnostics.
2. To restrict runs to or exclude specific calendar dates, use the optional **date filter** (see below).
3. Select **Save**. Save is explicit; nothing is scheduled until you save.

Restrict runs to specific dates (optional)

The date filter layers on top of the cron occurrences, it never changes the cron itself.

- Choose **Whitelist** to run **only** on the listed dates, or **Blacklist** to run on all scheduled dates **except** those listed. A common pattern is a weekday cron plus a Blacklist of holidays.
- Enter dates as a plain-text list (one per line) or upload a `.txt` file.
- Use slash-delimited dates: a single date as `yyyy/MM/dd`, and a range as `yyyy/MM/dd-yyyy/MM/dd` (dash-delimited, inclusive). Do not use ISO `yyyy-MM-dd` for a single date; the dash is reserved for the range separator, and a dashed single date is skipped.

A safe way to test a schedule before it goes live

1. In the editor, turn **Hold for Delivery** on and clear **Notify on failures only** so you get the run summary.
2. Save, then from the schedule's row on **Scheduled Content** select **Run now**.
3. Review the notification email and the held output in [Activity](#).

4. When it looks right, edit the schedule and turn **Hold for Delivery** off for production.

Notes and limits

- **Run now** is a Scheduled Content list action, not an editor action; running requires a saved schedule.
- A per-schedule **enabled** toggle is separate from the global master scheduler. Both must be on for a schedule to run. See [Manage scheduled content](#).

Going deeper. To turn scheduling on or off for the whole workspace, understand missed-run (catch-up) behavior, and monitor schedule health, see [Manage scheduled content](#).

Run estimates and expected completion

explanation report author administrator

Reportworq tells you not just *that* a job is running, but roughly *how much longer it has to go*, and afterwards whether a finished run met the expectation it was held to at the time.

This shows up in three places:

- **A live estimate** on the running job's row in [Activity](#), a small chip next to the elapsed time reading something like `~20m left`.
- **A running-jobs preview**, from the activity pulse icon in the top navigation bar, so you can check on activity from any screen.
- **An Est. completion lane** on a finished run's diagnostics Timeline, showing how long that run was expected to take, drawn on the same clock as everything else it did.

All three are read-only, at-a-glance aids. **None of them changes how a job runs or when it finishes.** They only report.

The live estimate on a running job

While a job is running, its **Running** row on the Activity screen shows an estimate chip beside the **Elapsed** column. It reads one of three ways, depending on where the run is relative to how long that job usually takes:

Chip	What it means
<code>~20m left</code>	The run is inside its usual envelope, and roughly this much time remains before its typical finish. At the very start of a run this reads as the whole estimate, for example "runs in about 20 minutes".
<code>wrapping up</code>	The run is right around its usual finishing time and should complete shortly.
<code>12m over</code>	The run has already been going longer than usual, by about this much.

Hover the chip for a fuller explanation, such as *"About 20 minutes remaining (usually ~47m, median of 8 similar runs)."*

`over` is informational, not an error. A run can legitimately take longer than usual, because a larger dataset or a slower datasource day will push it out. Treat it as an early heads-up worth a look.

How the estimate is calculated

The estimate is the **typical (median) duration of the job's own recent comparable runs**.

Reportworq uses the median rather than the average deliberately, so that a single unusual run, a one-off datasource timeout or a burst that was ten times its normal size, cannot distort the estimate for every run after it.

A past run only counts toward the estimate when it is genuinely comparable to the one in progress:

- it is the **same job**, in the **same workspace**;
- it **succeeded**, because a failed or canceled run stopped early for reasons unrelated to performance; and
- it has the same **shape**. A job run on its own is compared only against other solo runs, never against times when it ran as part of a larger scheduled batch, whose elapsed time covers every report in the batch.

Because the estimate is drawn only from the running job's own workspace, a similarly named job in a different workspace never affects it, and you only ever see estimates built from runs you are permitted to see.

When no estimate appears

If a job does not yet have **enough comparable history**, no chip is shown and the row simply displays its elapsed time. A job typically needs at least **three** comparable successful runs before an estimate appears.

This is deliberate. Reportworq shows nothing rather than a guess it cannot stand behind, so a chip that *is* shown is one you can rely on.

That also answers the common question of why one running job shows an estimate and another does not: the one without a chip is either new, or its recent runs do not match the current run's shape, for example it usually runs inside a batch but is running on its own this time.

The running-jobs preview

The **activity pulse** icon (the wave pulse) in the top navigation bar previews what is running right now. **Hover** it, or **tab** to it if you navigate by keyboard, and a small popover lists every currently running job with its name, elapsed time, current status, and its run estimate where one is available.

When nothing is running, the popover reads "**No jobs running right now.**" Clicking the pulse still opens the full Activity screen, as before.

Three things worth knowing:

- The preview lists only jobs **you are permitted to see**. It respects your workspace and folder access, so it never reveals a job you could not otherwise open.
- It opens on both mouse hover and keyboard focus, so it is reachable without a mouse. It contains no buttons, it is purely informational.
- On the Activity screen itself the popover is intentionally **not** shown, because you are already looking at the live list. Everywhere else in the app, it is available.

For full detail, per-run actions, and history, open the [Activity](#) screen.

Expected completion on a finished run

Open a run's diagnostics and choose the **Timeline** view. Its bottom lane, **Est. completion**, shows what that run was expected to take:

- an **expected band** running from the start of the run to the point it was expected to finish, labeled with the duration, for example `Expected 10.5s` ;
- when the run went **longer** than expected, a hatched **overrun** strip covering the stretch from the expected finish to where the run actually ended, labeled with the amount, for example `487ms over` ;
- when the run **beat** its estimate, the band runs to the right-hand edge of the timeline and fades out. The expected finish is off the end of the chart because the run stopped before reaching it, and the band's label says so: `Expected 10.8s · 426ms early` .

Hover the band for the full explanation, including how many past runs it was formed from.

The fading band does not mean data is missing. The fade means the expected finish is beyond the right-hand edge of the chart. The timeline's axis is always the length of the *actual* run, and is never stretched to fit an estimate, because that would squash the run's real activity into a corner of the plot.

Because the lane shares the timeline's one clock with the execution, workbook, event and memory/CPU lanes, you can see not just *that* a run went long but **what it was doing** when it passed the point it should have finished.

A finished run is judged by its own era

This is the important difference from the live chip. A finished run is compared against **the runs that came before it**, never against runs that happened afterwards. A run from March is judged by what Reportworq expected of it in March.

That matters because the alternative is subtly misleading. Comparing an old run against today's history would hold it to an expectation that did not exist yet, and the resulting number would look entirely plausible. As a job gets faster or slower over the months, older runs keep the expectations of their own era rather than being re-judged against today's.

It is the same rule the **Elapsed vs baseline** column on the run list uses, and the two are computed from the same place, so **they cannot disagree**. If the run list shows no comparison for a run, the timeline lane shows no band either, and both give the same reason.

When no band appears

The lane always appears, but it shows no band, and says why in plain words, when the run cannot honestly be compared:

What the lane says	Why
<i>"First run with telemetry, nothing to compare against yet."</i>	Nothing preceded it.
<i>"Only 2 comparable runs so far; a baseline needs 3."</i>	Too little history. Reportworq will not build an expectation from one or two prior runs, because ordinary variation would masquerade as normal.
<i>"This run failed, its duration isn't comparable."</i>	A failed run stopped early for reasons unrelated to performance.
<i>"This run was canceled, its duration isn't comparable."</i>	As above.
<i>"This run recorded no performance telemetry."</i>	Runs from before job telemetry existed, and held-report releases, have nothing to measure.

Who sees this

The expected-completion lane lives inside a run's **performance diagnostics**, which are gated the same way as the run list's comparative columns. A reader sees their own run's elapsed time, not the installation's performance diagnostics. This is deliberate.

Related

- [Activity and run history](#) for the screen the live estimate appears on.
- [Run performance](#) for the per-run breakdown the Timeline sits beside.
- [Find when a report got slower](#) for comparing runs over time.

Author a contribution campaign

[how-to](#)[report author](#)

A **campaign** is the object that runs a write-back cycle: it defines the input forms people fill, who fills and approves them, the notifications sent along the way, and the export that pushes collected data to a target system. You build it in the **campaign editor**, a left rail of five sections.

The worked example is a **cost-center budget round**: one budget form per cost center, each routed to its owner, reviewed by finance, and written back to a Planning Analytics cube on approval.

Before you start

- You can build a report. A campaign reuses the same source-worksheets-and-parameters surface as any Reportworq report.
- Your form worksheets carry the contribution formulas that define input cells, validation, and controls. See [Author the form worksheets](#) below, and [Contribution functions](#) for the full reference.
- Contacts and groups exist in the address book for the people you will assign.
- For write-back to reach your plan, the **Contribution** capability (which includes Data Collection) is licensed and an export destination is configured for your instance.

The five sections

The editor rail runs in order: **Create Forms**, **Assign Forms**, **Notifications**, **Exports**, **Options**. Each carries a count badge.

1. **Create Forms** authors the form content. It reuses the job editor, with child steps **Reports**, **Parameters**, **Bursting**, **Options**, and **Test**. Pick your source worksheets and map parameters here. See [How one definition becomes many forms](#).
2. **Assign Forms** generates the forms and routes them to people.
3. **Notifications** holds the built-in message templates for the cycle.
4. **Exports** configures where collected data is written back.
5. **Options** holds the campaign-wide settings, including approvals.

Rename the campaign inline from the editor chrome. Entering edit mode shows an **Unsaved** state with **Save** and **Discard**; a brand-new, unedited campaign opens clean with Save disabled.

How one definition becomes many forms

Generation produces **one form for every output the report definition would produce**, so the ways a report fans out are exactly the ways forms fan out. **A burst parameter is not required for a report to be replicated in a campaign.** There are two independent mechanisms, and you can use either, both, or neither.

- **One report per item.** On the **Parameters** step, open a parameter and set its replication to **One report per item**. Every value the parameter resolves to then produces its own report, and therefore its own form. This is the simplest way to get one form per cost center. Note the contrast with **One page per item**, which keeps every value in a single workbook and so produces one form, not many.
- **Burst sets.** Select **Enable this parameter to be used in burst sets** on a parameter, then build the burst sets on the **Bursting** step. Each burst set entry produces its own form and can carry a contact. Use this when the list of values and the people behind them are maintained as a set rather than derived from a query.

Using both multiplies them: every burst set entry is generated for each combination of the one-report-per-item values. Using neither produces a single form for the whole campaign.

For the cost-center budget round, set the **Cost Center** parameter to **One report per item** on the Parameters step. Generation then produces one budget form per cost center, with no burst parameter anywhere in the campaign.

Author the form worksheets

The forms are only as capable as the workbook behind them. At generation time Reportworq reads the **RW.Contribution** formulas in the source worksheets and renders the matching controls in the browser, so a worksheet with none of them generates a form no one can enter values into. Three things are load-bearing for a campaign.

- **RW.Contribution.Input** makes cells editable, and names the data model. Without it there are no input cells. Its second argument is the form (data model) name, and the repeating `fieldName, fieldReference` pairs that follow are that model's fields. This is not just labeling: the **Data Model** an export requires is exactly the name you put in this argument, and the `Fields.*` tokens you map to cube dimensions or table columns are exactly these field names. Choose them deliberately, because the export is built on them.
- **Keep one data model consistent across forms.** If two forms declare the same data model name with a different set of field names, the export cannot resolve the model and reports that the data model is not consistent across forms. Use one

`RW.Contribution.Input` pattern for all the forms a single export will cover.

- `RW.Contribution.Check` **decides what a contributor is allowed to submit.** A check marked as required blocks Submit until it passes, so it is the control that keeps bad values out of the plan before anyone approves them. A check that is not required warns without blocking.

`RW.Contribution.Checkbox`, `RW.Contribution.List`, and `RW.Contribution.Filter` are optional. They add a toggle, a controlled dropdown, and a row filter, and they are worth adding when free typing would let contributors invent values you then have to clean up.

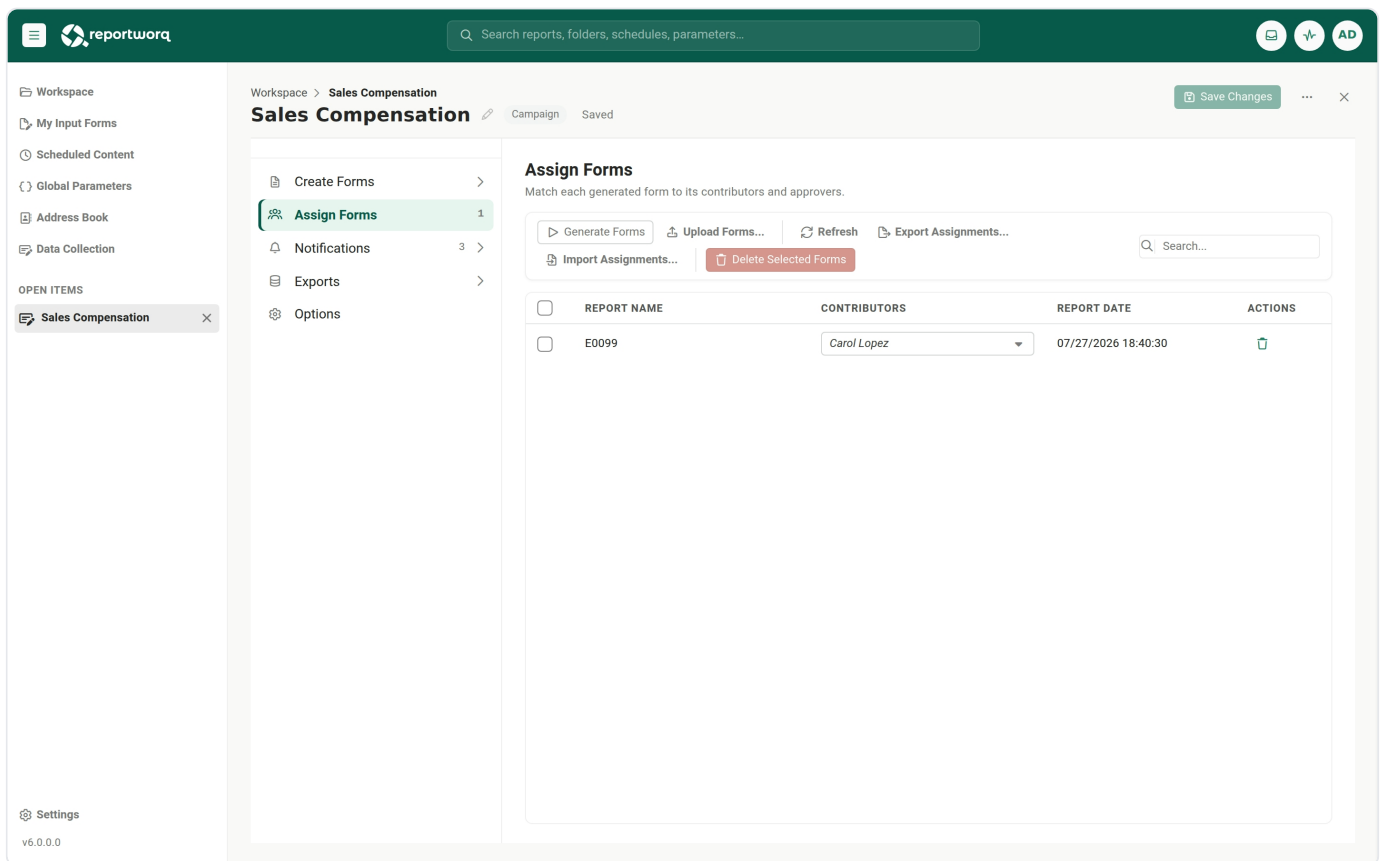
Two placement rules break more forms than anything else: never place an input, validation, or filter formula on a header row or header column, and never let an input range overlap a control cell. Full signatures, defaults, and the workbook design rules are in [Contribution functions](#).

Going deeper. A campaign's own export writes collected data to the target through the field mapping described above. The separate **data collection** path, where a workbook is returned by upload or mailbox and posts its values through hidden write-back formulas, is a different process. See [Contribution \(write-back\) overview](#).

Generate and assign forms

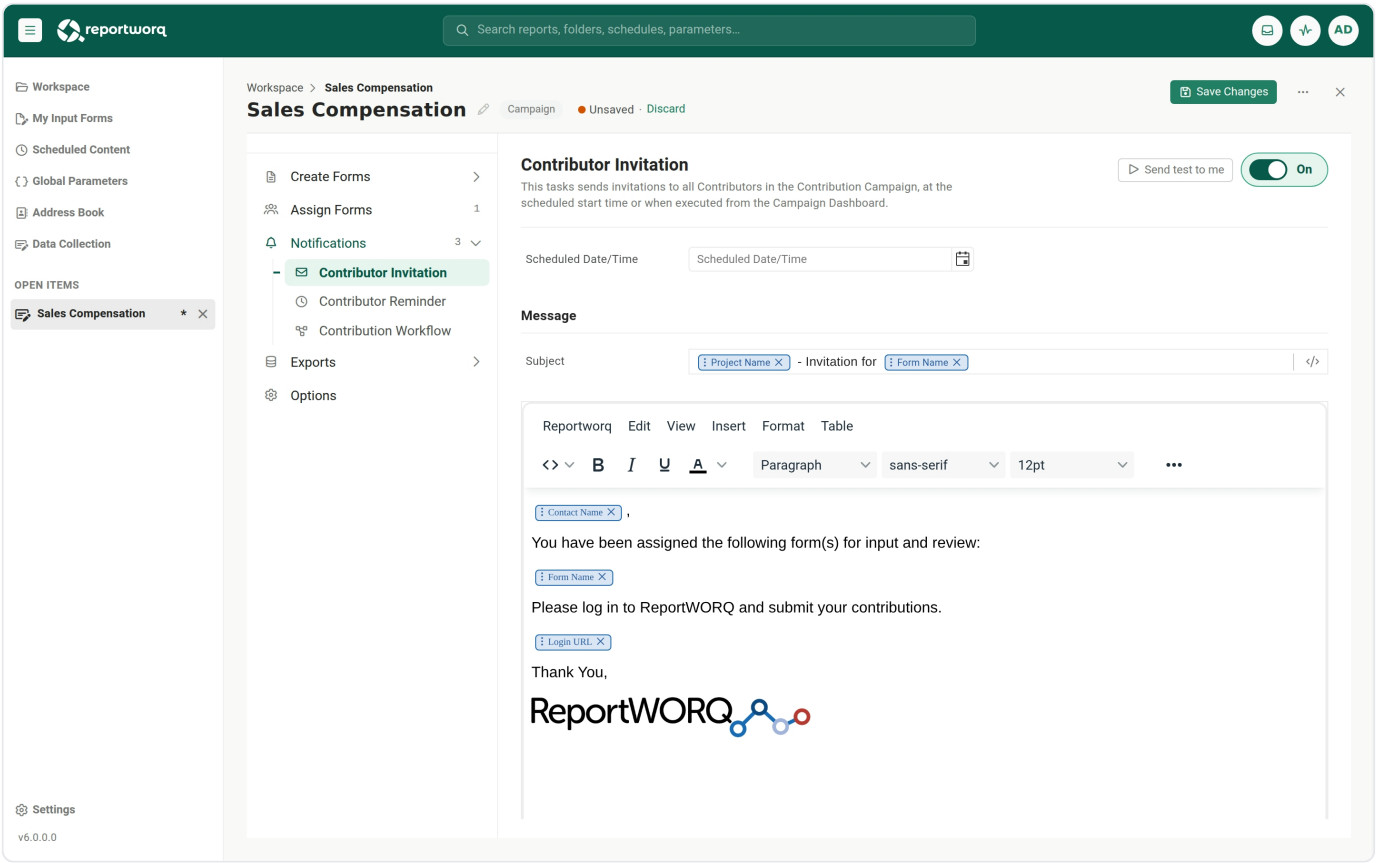
1. Open **Assign Forms** and select **Generate Forms**. This turns the Create Forms definition into concrete forms, one per output the definition produces.
2. In the assignment grid, set each form's **Contributors** (who fills it) from the multi-select of accounts and groups.
3. If approvals are on, set each form's **Approvers**. The Approvers column appears only when approvals are required.
4. For large rosters, use **Export Assignments...** and **Import Assignments...** to edit the mapping in bulk instead of row by row.

Assigned forms then appear to each person in [My Input Forms](#).



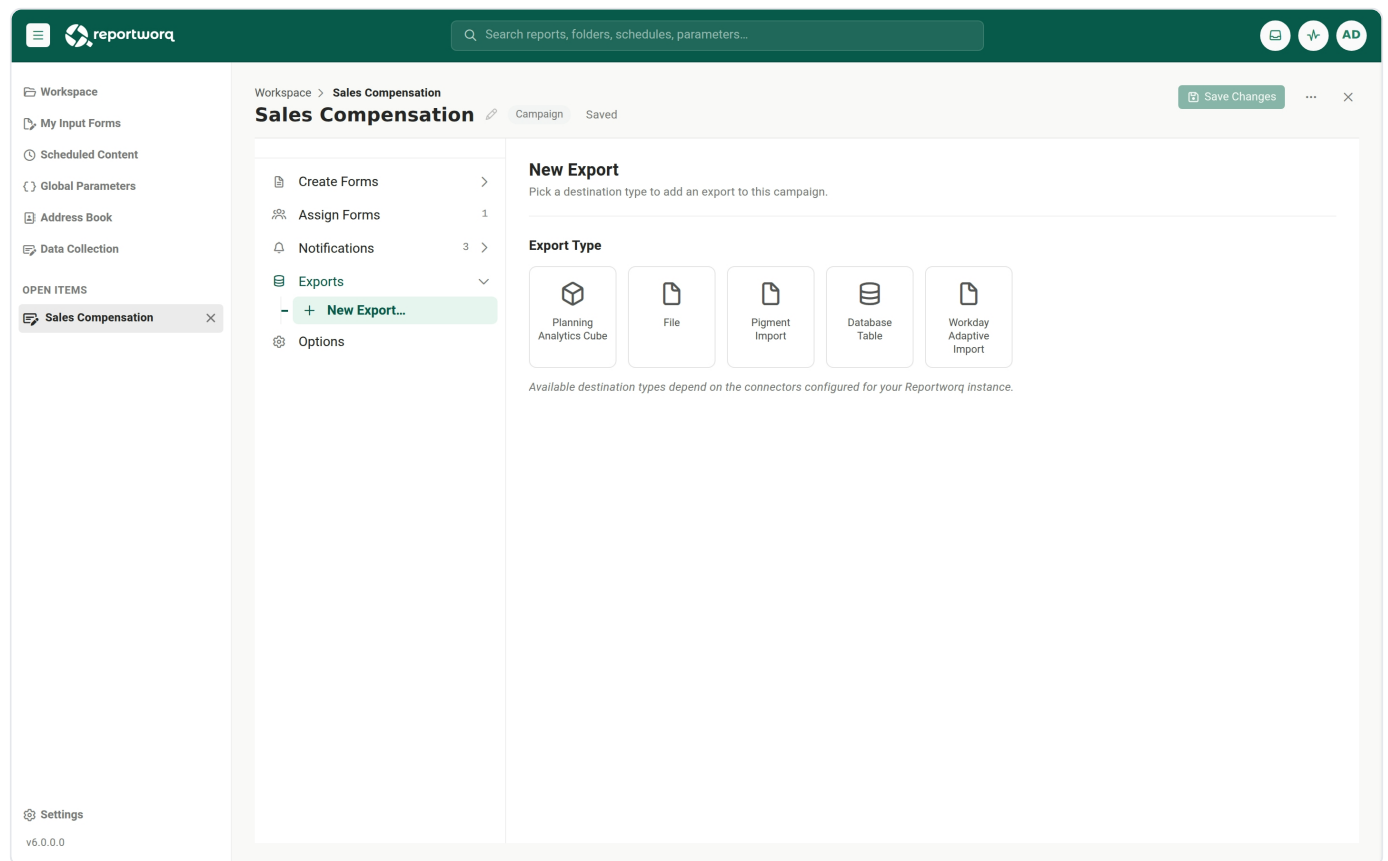
Regenerating forms. Regenerate Input Forms, on the campaign dashboard, rebuilds the generated forms from the current definition with refreshed source data. **Refresh Forms** is the equivalent trigger action for automation, and Auto Archive can **regenerate input forms after archiving**. A contributor's **Reset Form** is different: it reverts one form to baseline and deletes that form's collected data. Coordinate any regeneration with in-progress submissions.

Set notifications



The **Notifications** section lists built-in templates you can edit. With approvals off, three contributor templates appear: **Invitation**, **Reminder**, and **Workflow**. Turning approvals on adds three approver templates, for six total. Each opens a slim template editor. Notifications fire from triggers or can be sent by hand from the dashboard.

Configure the export (write-back)



The **Exports** section is where collected data is pushed to the target. Select **New Export...** to pick a destination tile, then configure it.

- **Export Model Type** chooses what is exported: **Data** (the input values), **Commentary** (contributor and approver comments), or **Workflow** (workflow activity per form). Data and Commentary require a **Data Model** name; Workflow needs none. Choosing Data or Commentary without a data model is invalid and the export will not run. The data model names on offer are the form names declared in the worksheets' `RW.Contribution.Input` formulas, and the `Fields.*` tokens you map to the destination are that formula's field names. See [Author the form worksheets](#).
- **Filters** (Data only) combine two choices into four options: approved forms only versus submitted-and-approved forms, and all data versus changes only.
- **Export Type (destination)** is a tile picker: **Planning Analytics Cube**, **Database Table**, or **File** (Excel `.xlsx`, or CSV with a Comma or Tab delimiter). Available tiles depend on the connectors configured for your instance.

An export can be run from the dashboard, or downloaded as an Excel workbook for review without writing to the target. Download is disabled until the export is valid.

Two behaviors matter when you rely on an export:

- **Work-in-progress is never exported.** An export always draws from approved (or, with the submitted-and-approved filter, submitted) forms. Draft values in an unsubmitted form are excluded under every filter.
- **A failed export reports "Has Warnings," not failure.** The export retries three times with a five-second wait; if all attempts fail, the activity is logged as *Warnings* ("Export Has Warnings"), not a hard error. A run that looks like it succeeded may have written nothing, so check the activity detail.

Set campaign options

The **Options** section (Campaign Options) holds the campaign-wide settings. The key ones:

- **Require approvals for submitted forms.** Turning this on re-renders Notifications to six templates and adds the Approvers column in Assign Forms. **One approval closes a form:** a submitted form needs a single approval regardless of how many approvers are assigned. Assigning several approvers spreads the ability to approve, it does not require a quorum.
- **Automatically stop campaign** and **Automatically archive campaign** each pair a toggle with an **Edit Schedule...** cron dialog. Auto Archive can also **regenerate input forms after archiving** and stop the campaign after archiving.
- **Contributor Export/Import options** govern the workbook a contributor can export or import: allow export/import, hide cells outside the print area, **restrict editing to input cells only**, and a **protection password**.

The **protection password is inert on its own.** The field is disabled unless *Restrict editing to input cells only* is on, and it protects the exported workbook only when that lock is enabled. Setting a password without the restrict lock protects nothing.

- **Manual calculation mode** lets contributors press **Recalculate** instead of recalculating on every edit, which helps large forms stay responsive.

Automate the cycle (triggers and actions)

A campaign can run on its own by pairing a **trigger** (when to fire) with an **action** (what to do).

Trigger	Fires when
Schedule	A cron schedule elapses (recurring or single run).
Form status	Selected workflow events occur: Refreshed, Approved, Rejected, Submitted, Submission Recalled, Approval Recalled.
Campaign status	The campaign starts running, is paused, or is stopped.
Webhook	An external caller POSTs to the campaign's generated endpoint.

Actions include **Export**, **Notification**, **Stop Campaign**, **Archive Campaign**, and **Refresh Forms**. A common setup: a form-status trigger exports each form to the cube on **Approved**, a schedule trigger stops submissions on the deadline, and Auto Archive archives the cycle when it ends.

Monitor the round from the campaign dashboard

Opening a running campaign shows its **dashboard**: one entry per input form, with the status filters and the bulk actions above them. This is where you watch a round progress and chase what has not come in.

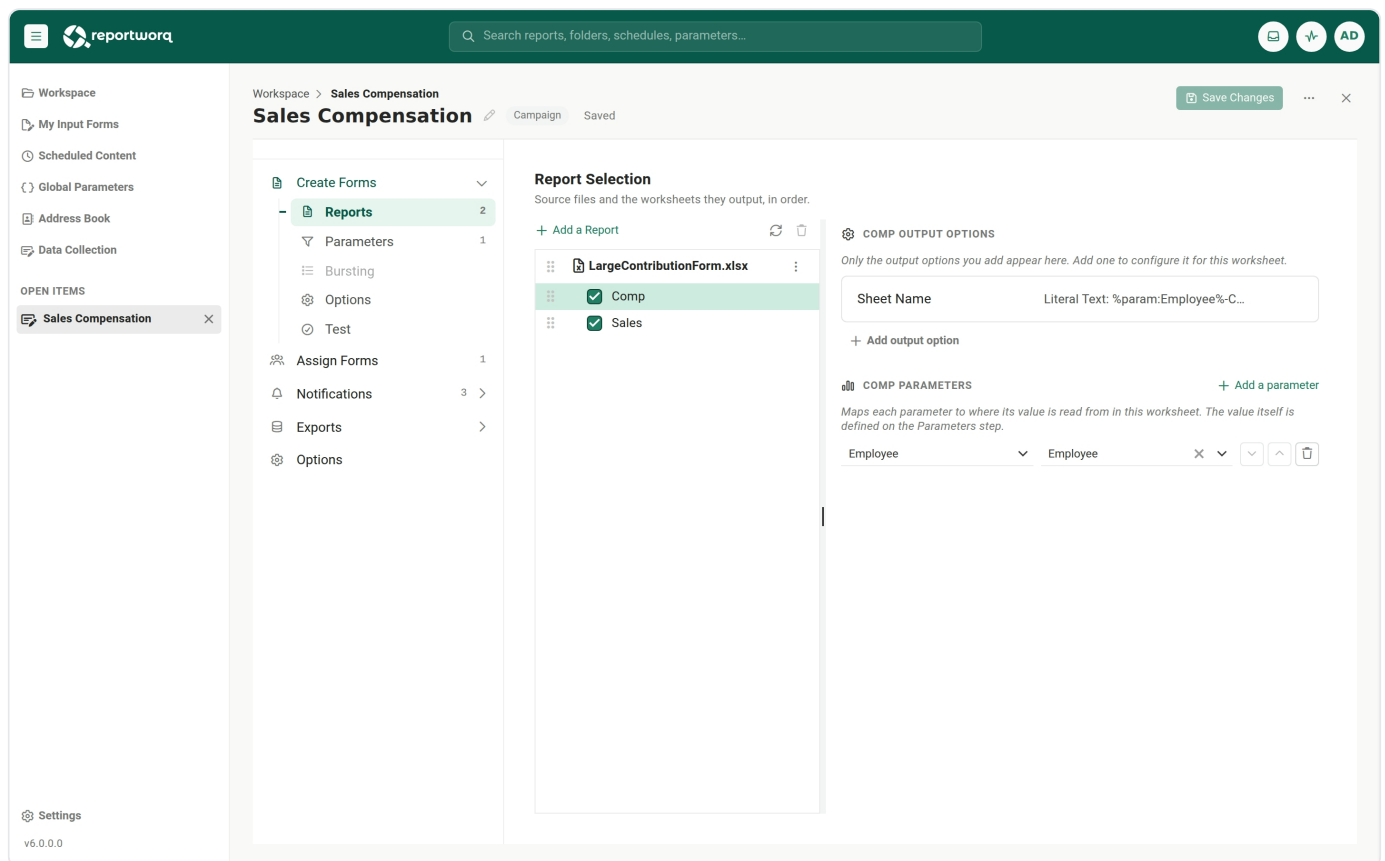
- **Filter by status.** The filters across the top are **All**, **Not Started**, **Rejected**, **In Progress**, **Submitted**, and **Approved**, each carrying a live count for the campaign. Selecting one narrows the list. Treat the counts as an indication of where the round stands rather than as a set of totals that must add up to All, because a form can satisfy more than one of them.
- **Act on a selection.** Select the forms you want with their checkboxes. A **N selected** badge appears with a **Clear** action beside it, and each bulk action's tooltip names what it will act on, either the selected forms or all forms in the campaign when nothing is selected. The badge stays visible as you page and filter, so an action can never quietly cover forms you have lost sight of.
- **Page through the forms.** A large campaign's forms are paged, and the selection survives paging.
- **Read the activity trail.** Open **Activity** from the campaign for the log of what happened to each form: when it was generated, notified, submitted, approved, rejected, or exported. **Export to Excel** saves the campaign's activity as `<CampaignName>_Activity.xlsx`, one row per entry with its **Created** time, **Task Name**, **Category**, **Status**, **Owner**, **Form Name**, and **Description**. Use it for an audit record of a completed round, or to chase stragglers in a spreadsheet. **Clear All Activity** empties the trail.

Managing a campaign after it runs

The campaigns list carries a per-campaign **Actions** menu, separate from editing. Two of its actions are easy to confuse, and they are not the same:

- **Archive Campaign Data** clears the working forms, activity, and collected data so the campaign can restart from a clean state, while keeping an archived record of the prior cycle.
- **Export Campaign** saves the whole campaign (its definition and its collected data) as a `.zip` to your Downloads folder, for backup or transfer to another instance. Its counterpart is **Import**.

Treating Archive Campaign Data as "save the campaign" would lose your working data. Use **Export Campaign** for a portable copy.



Notes and limits

- A **burst parameter is not required** for a report to be replicated into many forms. A parameter set to **One report per item** does the same job. See [How one definition becomes many forms](#).
- The **Bursting** child under Create Forms stays disabled until a parameter is enabled for burst sets. That gate applies to burst sets only, not to form replication in general.

- **Data/Commentary exports require a Data Model name;** without it the export is invalid. The available names come from the `RW.Contribution.Input` formulas in the form worksheets.
- **A data model must be declared identically across the forms one export covers.** Mismatched field lists under the same model name make the export unresolvable.
- **Regeneration rebuilds forms from the current definition.** Coordinate **Regenerate Input Forms** with anyone mid-submission.
- Write-back to a target needs the licensed **Contribution** capability, which includes Data Collection.

Going deeper. Data collection, the separate path where returned workbooks are processed from an upload or a monitored mailbox, has its own configuration and activity view. See [Data collection processing](#).

Related topics

- [Contribution functions](#)
- [Scale and size a contribution campaign](#)
- [Fill in and submit a form](#)

The report lifecycle

explanation

report author

user

The best way to understand Reportworq is to start from the end and work backwards, the same way a demo does. Picture what a store manager receives every Monday morning: a personalized email with a short chart and a few AI-written insights on their numbers, and a PDF attached with a table of contents, all specific to their store. Now rewind. How did that get built, and how does it keep arriving every week without anyone touching it?

This page follows one report, a weekly profit-and-loss pack sent to a retailer's stores, through its whole lifecycle: **author, parameterize, test, distribute, schedule, monitor**. The point is not the individual screens (each has its own how-to) but the shape of the journey.

The starting point: one solid template

The retailer has 150 stores. Each store manager needs the same statement with their own store's numbers. The instinct is to build 150 files. The aha moment of Reportworq is that you do not: you build **one** solid source template in Excel, and Reportworq iterates, slices, and personalizes it from there. If one manager needs their view shown a little differently, that is not a new template; Reportworq can hide rows, drop columns, and construct the variation within the tool. One master template, not a folder of near-duplicates.

Author

Authoring is where you define the report, the template that makes the outputs. You point the report at the source workbook and choose which worksheets it renders, in order. This is the Reports step of the Job Editor and the subject of [Your first report, end to end](#).

At this stage the report exists but is not yet personalized, it would produce the same statement for everyone. The next step is what makes it fan out.

Parameterize

Parameterizing is the most important idea in the lifecycle. You build the source file so that changing a single cell, say a **Store** cell from "Store 001" to "Store 002", causes every formula in the workbook to recalculate for that store. You then expose that cell to Reportworq as a **parameter**.

Now the report is a machine: feed it a store number and it produces that store's statement. Feed it a list of store numbers and it produces one per store. The care you spend here is what lets a single template serve 150 audiences. There is one rule worth knowing early: **map the parameter to every worksheet that should replicate with it**, even where a worksheet does not display the value. Do that and Reportworq copies those worksheets as a group and rewrites their cross-sheet references to the sibling copies, so a second tab that reads from the first stays correct in every copy. Map it to only one tab and the others will not replicate with it. The details, and the cell, pivot, and specification mappings you can use, are in [Parameterize a report](#).

Test

Before anything goes out, you preview. The **Test** step runs the report through the exact production pipeline but suppresses delivery, so nothing is emailed or written to a destination. You generate a preview, run it, download the produced file, and read the log (for example *Success - 1 datapoints*) to confirm the data resolved and the layout is right. It is your safe dry run: source binding, parameter mapping, and formatting are all exercised without touching a recipient. You saw this step in the [first-report tutorial](#).

Distribute

Distribution is the last mile, getting the trusted content to where people already work. This is where two things commonly surprise new users:

- **One job can produce many formats to many places.** You do not need a separate job for the PDF, another for the PowerPoint, and another for the email. A single job can render several output formats and send them to several destinations (email, network folder, SharePoint, Slack, Teams, and more) at once. See [Distribution overview](#) and [Output formats](#).
- **You usually do not need the bursting screen to burst.** Most people assume they must hand-maintain a list of who gets what. In practice you set a parameter to produce one report per item and let its values come straight from your source system, for example an IBM Planning Analytics subset or an attribute on each store, so the output follows the source data and updates itself when the business changes. That is replication, and it needs no burst sets and no **Burstable** flag. Burst sets exist for the cases replication cannot express, but they are the exception, not the rule. See [What bursting is and when you need it](#).

For the retailer, this is the step that turns one template into 150 personalized emails, each with the right PDF, without a per-store list to maintain.

Schedule

A report that is ready to run is a report that can run itself. Scheduling is the most underused capability in Reportworq, often because an author is not yet confident the job is dynamic enough to run unattended, which is exactly what the earlier steps build confidence in. Once the weekly P&L is parameterized and tested, you put it on a schedule (Reportworq uses a five-field cron expression) and it delivers every Monday with no manual step.

Scheduling does not have to mean "send automatically." You can schedule a run to **hold for delivery**, which produces the outputs on cadence but waits for a person to review and release them, the recurring equivalent of running a test and checking it before it goes. See [Schedule a job](#), or run it once by hand with [Run a job](#).

Monitor

Once a report runs on its own, you watch it rather than drive it. The monitoring surfaces show what is running, what is held for release, and the history of past runs, per report, so you can confirm the Monday send went out, see where its outputs landed, and release anything held for review. When a run fails, the log and the built-in diagnostics explain why. See [Track activity and run history](#) and [When a job fails](#).

The loop, not the line

Laid out in order the lifecycle looks linear, but in practice it is a loop. The template is defined once and reused every cycle. When a number changes at close, you do not rebuild anything; you rerun. When the business reorganizes, a query-driven report follows the new metadata on its own, no workbook edits, no distribution edits. That reuse, one authored template producing thousands of governed, personalized outputs on a schedule, is the whole point.

Where to go next

- Build the template in [Your first report, end to end](#).
- Make it personalize in [Parameterize a report](#).
- Send it in [Distribution overview](#).
- Automate it in [Schedule a job](#).

Item properties

[how-to](#)[report author](#)[administrator](#)

Every workspace item - a report, a contribution campaign, a data model, or a folder - has a **Properties** panel. It gathers the item's metadata and per-item settings in one place, so you can see where an item lives, who can reach it, whether it is scheduled, and how it is described for AI and search, all without opening the full editor.

Open Properties

From the workspace browser, open an item's row and choose the **info** action, or open the **⋮** menu and choose **Properties**. The panel replaces the browser; the **Workspace** breadcrumb (top left) closes it and returns you to the list.

Save and **Close** live in the top bar. There is no separate footer, and there is no separate Save button per field: editing any field marks the panel dirty and enables **Save**. If you try to leave with unsaved changes, Reportworq prompts you to discard them first.

What each tab holds

Which tabs appear depends on the item type and your role. A report shows the most; a folder shows fewer; a user with view-only access sees **Overview** alone.

Overview

- **Description** - a free-text description shown in the item table and in search results. Click it to edit; press **Ctrl+Enter** (or **⌘+Enter**) to commit, since a plain **Enter** starts a new line. The **title** is editable the same way, committed with **Enter**.
- **Retention** - how long this item's run history is kept:
 - **Inherit** - follow the folder chain, and ultimately the system default.
 - **Keep all history** - never removed automatically.
 - **Keep for a period** - a number of **days**, **months**, or **years**. A per-item choice overrides whatever the folder would otherwise apply. On a **folder**, this is the setting that cascades to the items inside it.
- **Visibility** - a **Hidden** switch that hides the item from readers while leaving it visible to authors and administrators. See [Roles and what you can do](#).
- **Details** - read-only **Path**, **Created**, and **Modified**, handy for confirming an item was moved to the right folder and recently updated.

Security

The **Security** tab shows the item's access-control list - who can see and act on this specific item, layered over the workspace's overall policy. Who may view and change it is a product rule, not a per-item permission:

- A **reader** (view-only) does not see the tab at all.
- An **author** sees it, but **read-only** - the panel shows a note that only an administrator can change who can see the content.
- An **administrator** can edit it.

For the full role model, see [Roles and what you can do](#).

Schedule

The **Schedule** tab lists the schedules that run this item. The list is read-only here: each schedule has an **edit** (pencil) action that opens it in the full Scheduler, and a **New schedule** action that opens the Scheduler to create one. All schedule authoring happens in the Scheduler, not inline. A schedule appears only when you can see every report it runs. See [Schedule a job](#).

AI Context

The **AI Context** tab holds descriptive metadata that helps the AI assistant and search decide when this item is relevant. It is related to, but separate from, the **AI Profile** you set inside the Job Editor. For authoring a report's AI context in depth, see [Create an AI Profile for a report](#).

Which tabs you see

Item and role	Tabs
Report or campaign, author or admin	Overview, Security (read-only for authors), Schedule, AI Context
Folder, author or admin	Overview, Security
Any item, view-only user	Overview (read-only)

Notes and limits

- Properties is for per-item governance - access, retention, visibility, schedule, and discoverability. The report's **content** (its source, parameters, bursting, and distribution) is edited in the Job Editor, not here. See [Choose the source workbook and worksheets](#).

- For a view-only user, Properties is often the only action available on an item's row, and it is read-only.

Parameterize a report

AUTH-02

how-to

report author

Parameterizing a report is the single most important idea in authoring. You build one solid source template in Excel, then drive it from named input values, called **parameters**, so that when you change a value, every formula in the workbook updates. Reportworq can then automate that change to distribute and burst a personalized copy for every period, entity, or recipient, without you maintaining a separate workbook for each.

The worked example is a **P&L** that goes out weekly to many stores. You want to change one "store" cell and have the whole report recalculate for that store.

Before you begin

- Add at least one source workbook first, on the [Reports step](#). A parameter is only useful once there is a worksheet to map it into.
- Confirm the workbook is built so that changing the driving cell actually updates the report. In your Excel file, change the store cell from one store to another and check that all the formulas recalculate. If they do not, fix the workbook before parameterizing. This check is what parameterizing depends on.

The order to work in

Most authors work **from the worksheet outwards**, and that is the flow this page follows:

1. **Start on the Reports step**, in the worksheet. Map a parameter to the cell or pivot field that drives it, on each worksheet that needs it.
2. **Then go to the Parameters step** and give each one its type and its values, or bind it to a query.

Working this way, you decide what the workbook needs while you are looking at the workbook, then say where the values come from. Starting on the Parameters step means declaring parameters before you know where they go, and then walking back to the Reports step anyway.

The exception, and the only reason to use Add Parameter on the Parameters step. Some parameters never appear on a worksheet at all, because they exist only for **distribution**: parameterizing an output file name, a folder path, or an email subject, perhaps fed from an API call that starts the job. A parameter like that has nothing to map, so create it directly on the Parameters step. If it belongs on a worksheet, add it from the Reports step instead.

Map a parameter to a worksheet

1. On the **Reports** step, open the worksheet's **Parameters** mapping panel.
2. Add the parameter and set its **Location**, the place its value is read.
3. Repeat for **every worksheet that needs the value**. This matters more than it looks; see the replication rule below.

Set the parameter's values

1. Open the **Parameters** step. The parameters you mapped are listed.
2. Select a row and **Edit** to open its sidebar. Choose a **Type** and give it values. **Text List** (one value per line) is the common starting choice; the full catalog is in the [Parameters reference](#).
3. Names must be unique within the report; reusing one raises a **Duplicate Parameter** alert.

reportworq

Search reports, folders, schedules, parameters...

AD

Workspace

My Input Forms

Scheduled Content

Global Parameters

Address Book

Data Collection

OPEN ITEMS

Commission Statements

Settings

v6.0.0.0

Workspace > Timberline > Financial Reports > Commission Statements

Commission Statements

Distribution job

Saved

Job Options

Save Changes

Reports

1 report · 4 sheets

Parameters

2 parameters

Bursting

Enable a burst parameter to fan this job out

Distribution

1 format · 1 destination

AI Profile

0 audiences · 0 scores

Test

Not run

Parameter Configuration

Default values, replication behavior, and which parameters can drive burst sets.

+ Add Parameter

NAME	MODE	DEFAULT VALUE	ACTIONS
Period			
Mapped in: Statement	One page per item	2026-May	
ReplID			
Mapped in: Statement	One report per item	E0105, E0168, E0169, E0107, E0213	

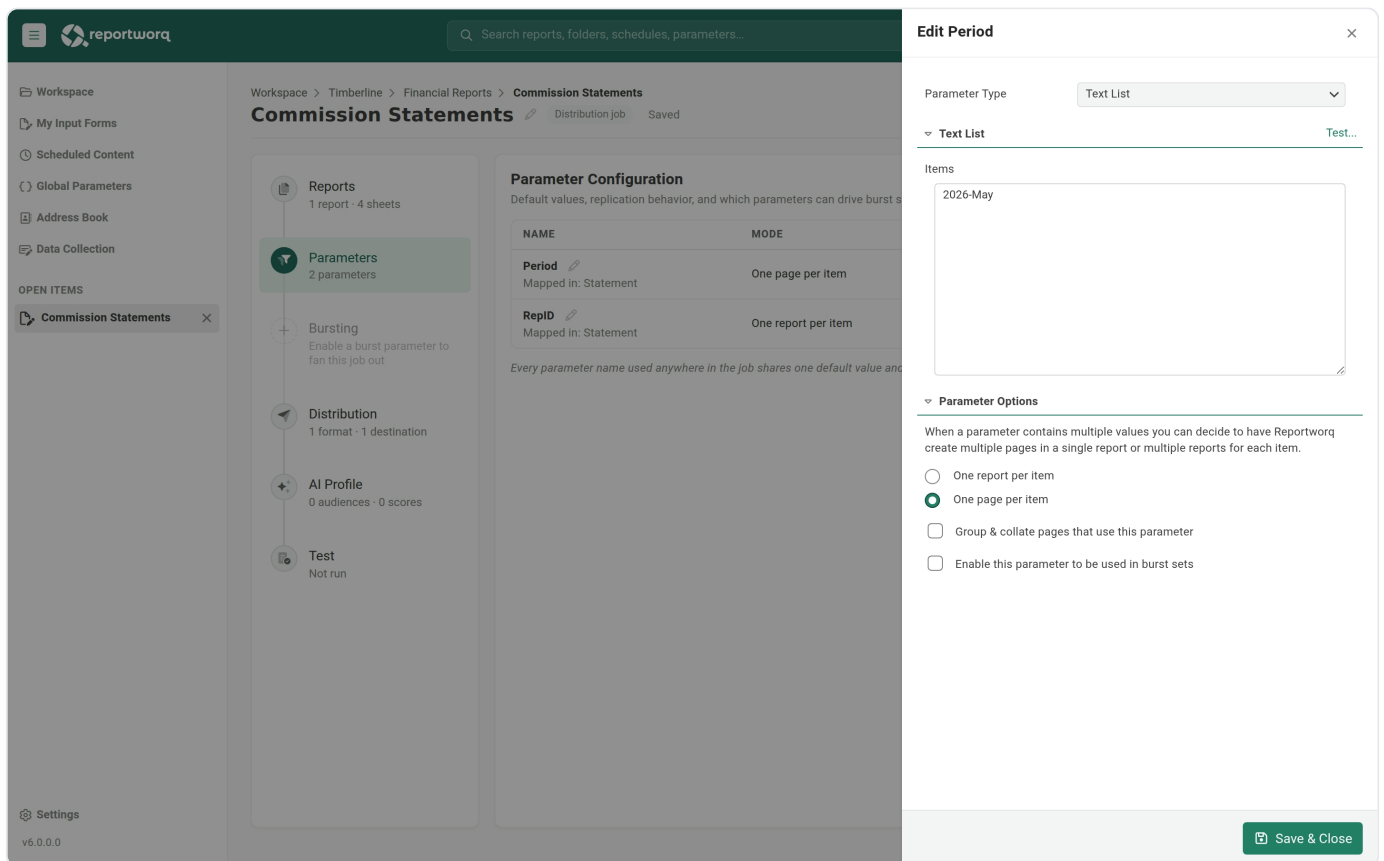
Every parameter name used anywhere in the job shares one default value and behavior. Flagging one as Burstable reveals the Bursting step.

Add Parameter

Enter Parameter Name

Cancel

OK



Where a parameter's value is read (the mapping)

A parameter's **Location** takes one of two shapes:

- **A cell or named range** (for example `B3`). The value is written there, so formulas, `VLOOKUP`, `RWSQL`, and everything downstream reads it. This is the common case.
- **A pivot-table field**, written as `PivotTableName.FieldName`. The value is applied as a pivot filter rather than written to a cell.

Use a named range, not a bare cell reference. `B3` is a position, and positions move. The day someone inserts a row above it, the parameter writes into the wrong cell and the report is quietly wrong rather than broken. A named range travels with the content it names.

What actually consumes the value

This is the idea worth internalizing: **Reportworq's job ends when the value lands in the cell.** It writes a value into a cell or applies it to a pivot field, and stops. What happens next is entirely up to how you built the workbook.

That loose coupling is the source of the product's flexibility. The parameter does not know what it drives, so it can drive anything Excel can express.

Worked examples

Filter a SQL query. Put `Region` into a named range, and write your query so the range supplies the `WHERE` clause, then pull the results with `=RWSQL`. Each replicated copy queries only its own region. Remember that text values need quoting in SQL:

`'{{region}}'` resolves to `'East'`, while numbers are bare. See [Connect SQL, OLE DB, and ODBC](#).

Drive a Planning Analytics tuple. Point a `DBRW` formula's element argument at the parameter's cell. The formula re-resolves per copy, so the same grid of `DBRW` calls reads a different slice of the cube for each entity, with no change to the formulas themselves.

Filter a Workday Adaptive report. Store the member selection with `RW.Adaptive.Filter` and have it read the parameter's cell, so `RW.Adaptive.Extract` builds each copy against a different filter. This is the preferred approach for Adaptive because it is **explicit**: the workbook shows what is being filtered. See [Advanced parameters](#).

Filter a replicating pivot table. Map the parameter to a pivot field rather than a cell. Each copy of the worksheet gets the pivot filtered to its own value, without a formula anywhere.

Look up who receives it. Put the cost center into a cell, then `VL00KUP` the email address and the Slack channel from a reference tab in the same workbook. Distribution reads those cells, so the same parameter that decides the *content* of a copy also decides *where it goes*. See [Distribution overview](#).

Feed an Excel add-in report. Point a Reportworq Excel add-in report at the parameter's cell as its filter value. The add-in re-queries per copy exactly as it would if you had typed the value yourself.

Name the output. Sheet-level functions such as `RW.Tools.SheetName` can read the same cell, so each generated worksheet names itself after the value that produced it.

In every one of these, Reportworq did the same small thing: it wrote a value. The workbook did the rest.

The replication rule that catches everyone

If a parameter replicates a worksheet, **map that parameter to every worksheet that must replicate with it**, even where the worksheet does not need to display the

value.

The classic case: a two-tab workbook where tab 2's formulas reference cells on tab 1, and you want one pair of tabs per store. **This does work**, as long as the store parameter is mapped on **both** tabs. Do that, and Reportworq copies the two worksheets **as a group**, and rewrites tab 2's cross-sheet references to point at its own sibling copy of tab 1 rather than at the original. Each pair is self-contained and correct.

Map it only on tab 1, and tab 1 replicates while tab 2 does not. Every copy of tab 2 keeps pointing at the original tab 1, and you get the same numbers repeated.

On tab 2 the parameter does not have to go anywhere visible. Map it to a spare cell if you have nothing to display; what matters is that Reportworq knows tab 2 belongs to the same replication.

Control tabs

The same rule explains control tabs, which are often described as an exception and are not.

- **A control tab that does not replicate** is entirely safe. Every worksheet pointing at it, replicating or not, keeps reading the one control tab, which is exactly what you want from a control tab.
- **A control tab that does replicate** is rewritten to siblings only for the worksheets replicating on the **same** parameter. Worksheets replicating on a different parameter, or not replicating at all, keep pointing at the original.

If that sounds familiar, it is the same behavior as copying worksheets in Excel by hand: copy a group of sheets together and their internal references follow the copies; copy one sheet on its own and it still points at the original.

Copying related worksheets as a group is **on by default**. An administrator can turn it off at **Settings > Configuration > Reporting > Advanced Features > Enable copying worksheets as a group**.

Going deeper. A parameter can reference another parameter's value and repeat a query once per item, fanning out one report per element. See [Advanced parameters](#). For the full type catalog, replication modes, and parameter sets, see the [Parameters reference](#).

Burst sets

[how-to](#)[report author](#)

A **burst set** is one row on the **Bursting** step, and it produces its own output. Each row carries a value for each burst parameter plus an optional recipient contact. Burst sets are the tool for **per-output control**: they let individual outputs differ from each other in ways a single replicating parameter cannot express.

When a burst set is worth it

Reach for a burst set when the outputs are not all built the same way:

- **Individual outputs need different values.** One output takes a hand-picked value, another takes a different query, and the rest take the standard one.
- **The list is fixed and curated**, with no home in a source system, such as a roster of stores or an external distribution list.
- **The list is easiest to maintain outside Reportworq**, in a spreadsheet the business already keeps or a standalone query.
- **Each output goes to its own recipient** and you want that pairing recorded alongside the values.

If instead every output is built identically and differs only by one value that already lives in your source system, a burst set is more work than you need. Set that parameter's **replication mode** on the **Parameters** step and it fans the output out on its own, with no Bursting step and no **Burstable** flag. See [Replicate output from your source system](#).

Before you start

- The job has at least one parameter, and each parameter you want to control per output has a type that can carry the values you intend to give it. See [Parameterize a report](#).
- For an Excel-sourced list, an accessible workbook in a configured report provider. For a SQL-sourced list, a configured SQL datasource connection.
- Recipients are optional. If you want them, see [Assign recipients to bursts](#).

Unlock the Bursting step

The **Bursting** step is hidden until at least one parameter is flagged **Burstable**. Revealing that step is the only thing the flag does, and building burst sets is the only reason to turn it on.

1. Open the job and go to the **Parameters** step.

2. Find the parameter you want a column for and open its **Edit** sidebar.
3. In **Parameter Options**, turn on **Enable this parameter to be used in burst sets**.
4. Select **Save & Close**.

The parameter row now shows a **Burstable** chip and the **Bursting** step is available.

Save matters. The Edit Parameters sidebar commits only on **Save & Close**. If you close it with Escape or the X, the flag is discarded and the Bursting step stays disabled.

Repeat for each parameter you want to control per set. The Bursting grid shows one column per Burstable parameter.

Choose where the sets come from

On the **Bursting** step, the **Source** toggle offers three modes, with **Static** first:

- **Static** builds the sets by hand in a grid. The count is known at edit time, so the step shows an "**N output(s) will be produced**" pill.
- **Excel Source** reads the sets from a workbook.
- **SQL Source** generates the sets from a query.

Source here means only where the **list of burst sets** comes from. It has nothing to do with where the report gets its data. A job whose report reads Planning Analytics can take its burst sets from a SQL query, from a workbook, or from the hand-built grid, whichever is easiest to maintain.

Excel Source and SQL Source regenerate their sets on every run, so they have no output count at edit time. A Planning Analytics subset is not a fourth mode: it is a **parameter type**. In the Static grid, a cell for a subset-typed parameter opens the subset tree picker. In the Excel and SQL modes the preview cells are read-only, because the values come from the file or the query.

Build sets by hand

1. On the **Bursting** step, keep **Source** on **Static**.
2. Select **Add a set**. Each set is one numbered row, and each column is a Burstable parameter.
3. Select a cell to open the **Edit {parameter}** sidebar. It is the same editor the Parameters step uses, so the cell offers whatever the parameter's type offers: a text list, a subset tree, an MDX or SQL query, a dimension picker.

4. Optionally set the **Contact (optional)** column to route the set's output. See [Assign recipients to bursts](#).
5. Repeat **Add a set** for each output. To remove a set, use its row delete action and confirm.

Let a default carry, and override only what differs

Every cell starts by inheriting the value already set on the **Parameters** step. A cell shows one of three states:

- **Override**, the set's own explicit value, marked with a teal override bar.
- **Default** followed by the inherited value, meaning the set carries nothing of its own and falls through to the job parameter.
- **not specified...**, meaning neither a per-set value nor a job-level value exists.

This is what keeps a large grid manageable when a burst parameter is query-driven. If the parameter is a Planning Analytics MDX query or a SQL query, the query you write once on the **Parameters** step becomes the default across every burst set. A grid of 30 sets needs that query entered once, and you override only the handful of sets that need a different query or a fixed value.

The same holds with several burst parameters. Each column inherits its own parameter's value independently, so one column can sit entirely on a shared default query while another carries a different hand-set value in every row.

At run time each set applies only the values it actually carries. A set with every cell left at **Default** still produces an output, built from the job's own parameter values rather than an empty file.

Read sets from an Excel workbook

1. Set **Source** to **Excel Source**.
2. In the **Excel File** row, select **Edit...** and choose a workbook from a configured report provider. Row 1 must hold the parameter names, and each following row is one burst set. Include a **Contact** column if you want each output routed from the file.
3. Choosing a file runs the preview automatically. Use **Refresh Preview** to re-read the file after it changes.
4. If a cell should carry several values, set **Value List Delimiter** to the single character that separates them.

The preview grid is read-only because the values come from the file. If you see the warning *"Some parameters could not be found in the file,"* align the row 1 headers to the parameter names. Because the sets are re-read on every run, editing the workbook

changes the roster without touching the job.

Generate sets from a SQL query

1. Set **Source** to **SQL Source**.
2. Choose a **Connection** from your configured SQL datasource connections.
3. Enter the **Query**. Each result row is one burst set, and each result column maps to a Burstable parameter by name. Return a contact column if you want each output routed from the result.
4. Optionally set **Value List Delimiter** for multi-value cells.
5. Select **Refresh Preview** to run the query and fill the grid.

The preview is read-only. If you see *"Some parameters could not be found in the query,"* make the result column names match the parameter names. The query re-runs on every run, so the roster always reflects the database.

Import and export static sets

Static mode can round-trip its sets through an Excel file named `burstsets.xlsx`. This is for bulk-editing many sets or copying a list between jobs, and it is offered in **Static** mode only.

- **Export** downloads `burstsets.xlsx`. Row 1 holds the burst parameter names plus a **Contact** heading, each following row is a set, and the contact is written by name.
- **Import...** uploads a burst-set workbook and **replaces** the current static sets with its rows. It reads the first visible worksheet, treats row 1 as headers, splits multi-value cells on a comma, and resolves contacts by name against the address book.

Import overwrites rather than merges, so export first if you want to keep the current sets. For a list that should stay live, use **Excel Source** or **SQL Source** instead, which regenerate on every run rather than snapshotting into static sets.

Use cases

- **Curated store roster.** Twelve flagship stores, entered as static sets, produce the same twelve packets every run regardless of how many stores exist in the source system.
- **One exception in a standard set.** Thirty regional packs all inherit the same MDX query as their default, and two regions that need a restated hierarchy carry their own query as an override.
- **A distribution list the business maintains.** An FP&A team keeps a workbook of departments and their recipients, and Excel Source turns each row into a personalized output with no re-keying in Reportworq.

- **A roster driven by a system of record.** A SQL query against the HR database returns each department and its manager, so the roster tracks reorganizations automatically.
- **Cloning a burst list.** Export the sets from a working P&L job and import them into a new variance job that covers the same departments.

Notes and limits

- **Static sets are stored on the job**, so the count is known at edit time. **Excel** and **SQL** sets are re-resolved on every run and have no count until the run happens.
- **Row 1 headers (Excel) and result column names (SQL) must match the burst parameter names**, or those values are missed.
- **A Planning Analytics subset or MDX value inside a set is resolved live.** Its members are known only at run time, and the run needs Planning Analytics connectivity.
- **Burst sets and replication compose.** Reportworq works through the sets first, applies each set's values on top of the job's parameters, then replicates within that set. If a parameter still replicates inside a set, that set produces more than one output and the "**N output(s) will be produced**" pill undercounts.
- **A contact is optional on every set.** A set with no contact still produces its output and is delivered by the job's own destinations.

Going deeper. The Repository Metadata Report includes a **Static Burst Sets** worksheet, so an administrator can audit every job's static sets in one place. See [Repository Metadata Report](#).

Output formats

DIST-02

reference

report author

On the **Distribution** step, **Output formats** is a set of tiles that select which file type or types a report generates. Formats are **additive**: a single run can emit Excel, PDF, PowerPoint, and CSV together, one output file per selected format. Each selected format exposes a **per-format options** sidebar (opened from the edit control on the format card) that controls how the report renders to that file type. The base file name comes from the file-name field ([Name and organize output files](#)); the extension is set by the format.

The screenshot displays the 'reportworq' interface for configuring a 'Financial Statements' job. The left sidebar shows a navigation menu with 'Workspace', 'My Input Forms', 'Scheduled Content', 'Global Parameters', 'Address Book', 'Data Collection', and 'OPEN ITEMS' (Financial Statements). The main workspace shows the 'Financial Statements' job in the 'Distribution' step. The 'Distribution Options' section includes:

- File Name:** A field with tokens 'Year' and 'Market' followed by 'Financial Statements'.
- File Format(s):** A grid of format tiles: Excel (selected), PDF (selected), PowerPoint, CSV, and Files.
- Destination(s):** A grid of destination tiles: Email (selected), GitHub, Google Drive, Network Folder, OneDrive, and SharePoint.
- Format Options:** Two panels for 'Excel' and 'PDF'. The Excel panel has 'Remove Provider Formulas'. The PDF panel has 'Add Page Numbers', 'Add Table of Contents', 'Custom Table of Contents ...', and 'Low-Quality PDF'.
- Email Options:** A section for email delivery with 'Attachments' (All Formats, Excel, PDF), 'To' (Cell EmailTo), and 'Subject' (Year, Market, Financial Statements).

One job, many formats. You do not need a separate job for the PDF, another for the PowerPoint, and another for the email. Select every format tile the run should produce, then route each format to the destinations that want it (see [Destinations](#)). **Selecting zero formats is also valid**, which sends a message-only delivery (for example an HTML newsletter or an exception report) with no attachment.

The formats

Format	Extension	What it produces
Excel	<code>.xlsx</code> / <code>.xlsm</code>	The rendered workbook. <code>.xlsm</code> when <i>Save as .XLSM</i> is enabled.
PDF	<code>.pdf</code>	The workbook (or PowerPoint) rendered to PDF, with optional table of contents and page numbers.
PowerPoint	<code>.pptx</code>	A deck rendered from a PowerPoint template . Requires a template attached to the job.
CSV	<code>.csv</code>	Comma-separated values from the report's tabular content.
Files	<i>(keeps each file's own extension)</i>	A static list of extra attachments shipped alongside the report. See the note below.

AI-generated insights are not an output format. An AI insight is narrative text embedded into an email message, not a file the run produces, so it is documented with the message editor. See [Email messages and templates](#).

"Files" (Additional Files) attaches extra documents, and is never merged into the PDF

The **Files** tile does not render the report to a new format. It attaches a **static list of arbitrary files** (browsed from a report provider, or uploaded) to the delivery. Each one ships as its **own separate attachment** next to the report output. It is a fixed list picked at design time; there is **no** token or pill editor, so you **cannot** vary the attachment per parameter value at run time (for example "attach `<State>.pdf` where State is the burst parameter" is not supported). The files are also **never inserted or merged into the compiled report PDF**, the packet and the extra files travel side by side.

When to use Files. You have a fixed cover letter, a data-dictionary PDF, or a terms sheet that must accompany every run unchanged. Attach it once as an Additional File.

Dynamically linked source reports vary the workbook per run

Because the Files list is fixed, it is the wrong tool when the *content* has to change per parameter value. The mechanism for that is a **dynamically linked source report**: on the Reports step, a source file's **Dynamic Path** property accepts parameter tokens in the form `%param:Name%`, and Reportworq resolves the path beneath the provider root at run time. One job then reads a **different workbook per parameter combination**, for example:

Templates/%param:Region%/pnl-pack.xlsx

A parameter with several values resolves to several source files, so a burst can render genuinely different templates per business unit rather than the same template with different filters. Set it from the source file's **Properties**, using the **Insert Parameter** control next to the **Dynamic Path** field. A report using it shows a lightning-bolt icon in the Reports grid.

Dynamic Path is available for distribution reports only, not for contribution campaigns. For the full treatment, including the provider-root security boundary, see [Source report providers](#).

For the other case, an external page that must appear *inside* the compiled report PDF, the Files tile cannot help either: additional files travel beside the packet. Regenerate that content from a data source so Reportworq renders it as part of the report, or host the file (for example in SharePoint) and email a **dynamic link** to it using a variable in the URL, see [Email messages and templates](#).

Per-format options

Excel

Option	Values / default	Notes
Formula Options	Keep All Formulas / Remove All Formulas / Remove Provider Formulas / Remove Provider Formulas (except Input Formulas)	Controls which formulas remain live in the delivered workbook. Applied to the whole workbook after the report calculates.
Save as .XLSM	On / Off	Emits a macro-enabled .xlsm (keeping the VBA project) instead of .xlsx. Prerequisite: the job must have a single .xlsm source report.

Each option freezes a different set of formulas to the values they last calculated:

Formula option	What stays live	What is frozen to a value
Keep All Formulas	Everything.	Nothing.
Remove All Formulas	Nothing.	Every formula in the workbook.
Remove Provider Formulas	Ordinary Excel formulas, such as <code>SUM</code> , arithmetic, and lookups.	Every data-source formula, the retrieval functions that query IBM Planning Analytics, SQL, and the other connectors.
Remove Provider Formulas (except Input Formulas)	Ordinary Excel formulas and the input (write-back) formulas.	The remaining data-source retrieval formulas.

When to use Remove Provider Formulas. The recipient has no access to the source system. Removing the provider formulas replaces every data-source retrieval with the value it returned during the run, so the workbook opens cleanly for someone with no Planning Analytics or database credentials and no Reportworq add-in, while **all the Excel-based formulas keep working**. Totals, ratios, and any what-if arithmetic the recipient builds on top still recalculate. Compare this with **Remove All Formulas**, which freezes the Excel formulas as well and delivers a purely static workbook.

When to use Keep All Formulas. The recipient is an analyst who has access to the source system and will keep refreshing the workbook against live data.

When to use Remove Provider Formulas (except Input Formulas). The workbook is a submission template. The retrieval formulas are frozen so the numbers are stable, but the input formulas survive, so the recipient can still write values back.

When Excel lands on Google Drive. Excel output is auto-converted to a Google Sheet on delivery to a Google Drive destination.

PDF

Option	Values / default	Notes
Table of Contents (+ Custom Title)	On / Off ; title text	Adds a TOC with an optional custom title.
Page Numbers	On / Off	Adds page numbering.
Low quality PDF	On / Off	Produces a smaller, lower-resolution file.

When to use Low quality PDF. The pack is large and is going out by email, where a smaller attachment matters more than print fidelity.

PowerPoint

Option	Values / default	Notes
Template File	A selected <code>.pptx</code> template	The template the slides render from. Required , a run with no template fails.
Output as PDF	On / Off	Also renders the deck to PDF.
Fit To Width	On / Off	Scales content to slide width.
Sort Slides	On / Off	Orders the slides.
Image Format	Image type for rendered images	Format of images placed on slides.
PowerPoint Image PPI	Resolution (pixels per inch)	Resolution of rendered images.
Allow PowerPoint user to refresh output	On / Off	Lets a PowerPoint add-in recipient refresh the linked output.
Allow PowerPoint to run job	On / Off	Lets the add-in run the job with fresh datasource queries. Requires the refresh permission above.

See [PowerPoint reporting](#) for the authoring side.

CSV

CSV emits the report's tabular content with no per-format sidebar of its own. Complex charts and pivots do not carry across; use CSV for figures a downstream system will load.

A CSV holds one flat sheet of values, so a multi-tab report produces a CSV of the **first worksheet only**. To get one `.csv` per tab, turn on **Split Worksheets**, see [Combine emails, collate, and zip](#).

Email packaging options

The Email distributor adds options that package the generated files for delivery, rather than change how they render:

Option	Notes
Zip Files	Bundle the outputs into a single <code>.zip</code> .
Exclusion List	Semicolon-separated names or extensions dropped from the zip (only applies when Zip Files is on).
Combine email messages when possible	Merge outputs that share the same addressing into a single email.
Combine using this job's email details	Across a multi-job run, designates the one job whose subject and body the combined email uses.

These are covered, with the **Combine Key** field that steers the grouping, in [Combine emails](#), [collate](#), and [zip](#).

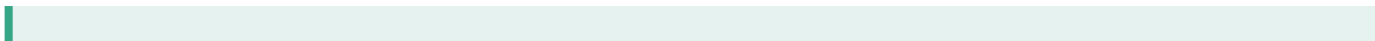
Use cases

- **Monthly P&L pack.** Excel (for analysts to pivot) *and* PDF (TOC plus page numbers, for controllers), generated in one run.
- **Board deck.** A PowerPoint template renders the presentation and *Output as PDF* produces a read-only companion.
- **Data extract.** CSV output loaded into a downstream system.
- **Workbook for a recipient with no source-system access.** Excel with **Remove Provider Formulas**, so the delivered file keeps its Excel formulas but no longer queries Planning Analytics or the database.
- **Per-region source templates.** A **Dynamic Path** on the source report reads a different workbook per region while one job drives the whole burst.

Notes and limits

- PowerPoint output requires a `.pptx` template attached to the job.
- Complex charts and pivots may not render faithfully in CSV, and a CSV covers the first worksheet only unless Split Worksheets is on.
- The **Files** (Additional Files) list is static and non-parameterizable, and its files ship as separate attachments, never merged into the report PDF.
- **Formula Options apply to the whole workbook**, after the report calculates. To freeze a specific range instead, use the `RWREMOVEFORMULA` worksheet function, see [General report functions](#). The two combine: the function runs during calculation, the Formula Option runs after it.

Going deeper. File names carry parameter, contact, and timestamp tokens so each generated file is distinctly named per burst iteration and recipient. See [Name and organize output files](#).



Excel: find and insert functions

[how-to](#)[report author](#)

Reportworq ships a library of **worksheet functions** you write into cells like any other Excel formula. You do not have to remember their names or their arguments. The **Formula Library** dock, pinned at the foot of the companion pane, is a searchable catalog that also writes the function into the sheet for you.

When to use it. Whenever you are writing a Reportworq function and want the signature, an example, and a correct skeleton in the cell rather than typing one from memory. It is also the fastest way to read a workbook someone else built.

Before you start

- The shared prerequisites in [The Office add-ins](#).
- The workbook open in Excel with the Reportworq task pane showing.

Find a function

1. Open the **Formula Library** dock at the foot of the pane. It expands into a near-full-height sheet.
2. Browse a family, or enter a term in the search box to filter across all of them.

The dock groups the catalog into three families:

- **Reporting** functions control the output itself: naming a sheet from a formula, coloring a tab, suppressing a page.
- **Data access** functions retrieve values from a connected system. These are grouped by connector family, so the Planning Analytics functions look different from the relational ones, and the Workday Adaptive ones different again.
- **Contribution** functions belong to write-back forms rather than read-only reports. See [Contribution functions](#).

Search does not require you to expand a family first. A query with no matches shows an empty state rather than an error.

Read what a function does

Select a function to open its **Learn** page. It carries the function's goal, its signature, each parameter, a worked example, best practices, related functions, and a link to the full documentation.

Function names have a long form and a short alias. `RW.Tools.SheetName` and `RWSHEETNAME` are the same function. **Prefer the long form in new work**, because it says which family the function belongs to, and it is the form the library and the reference use.

Insert a function into a cell

1. Select the cell on the worksheet where the function should go.
2. In the library, open the function you want.
3. Choose **Insert**.

Reportworq writes a skeleton call into the active cell and confirms with a toast. Fill in the arguments from the signature on the Learn page.

Open the Learn page for a function already in the sheet

Select a cell that already holds a Reportworq function, and the library opens **straight at that function's Learn page**. This works for both the long form and the short alias.

This is the quickest way to understand a workbook you inherited: select the cell, read what the function does. Selecting a cell that holds a plain value, or a native Excel formula, does not open the library.

Notes and limits

- The library **inserts a skeleton, not a working formula**. You still supply the arguments.
- **Insert targets the active cell**, so the cell you want has to be selected first.
- Reportworq functions are **inert in desktop Excel**. Nothing happens in the cell until the report runs. See [Functions overview](#).

Going deeper. For the full catalog by connector family, what each function does, and which one to reach for when, see [Functions overview](#).

The AI insight function

reference

report author

`RW.Tools.AIInsight` calls a configured AI provider with a prompt and writes the response directly into a named Excel TextBox or a target cell at report-generation time. Use it to embed AI-generated summaries, commentary or analysis into a report without any manual copy-and-paste step.

Alias: `RWAIINSIGHT`.

This is the only generative worksheet function. The `AI*`-aliased functions (`AIEXTRACT`, `AIINPUT` and the rest) are the [Workday Adaptive](#) connector, where "AI" stands for Adaptive Insights. They have nothing to do with large language models.

Prerequisites

An AI provider must be configured **and enabled** before this function does anything. See [Connect an AI provider](#).

RW.Tools.AIInsight

Signature

```
=RW.Tools.AIInsight(providerName, destination, prompt, [dataRange], [maxTokens])
```

#	Argument	Type	Default	Meaning
1	<code>providerName</code>	String	<i>required</i>	The Name or ProviderId of a configured AI provider, for example <code>"OpenAI"</code> . Lookup is case-insensitive.
2	<code>destination</code>	String or cell reference	<i>required</i>	Where the response is written. A string is treated as a TextBox name; a cell reference writes to a cell.
3	<code>prompt</code>	String	<i>required</i>	The prompt sent to the provider.
4	<code>dataRange</code>	Range	<i>none</i>	A range whose values are appended to the prompt as a Markdown table.
5	<code>maxTokens</code>	Integer	<code>1000</code>	The maximum output tokens the provider may return, itself capped by the provider's own configured limit.

The first three arguments are required.

How the prompt is built

The full prompt sent to the provider is assembled in two parts:

1. The literal text you supplied in `prompt`.
2. If `dataRange` is supplied, a blank line followed by that range rendered as a Markdown table.

That is the whole mechanism. There is no hidden system prompt added by the function itself, and no chunking: the range goes over as a table.

Where the response goes

A TextBox destination searches every worksheet in the workbook for a TextBox with that name, and replaces its text content with the response. If no TextBox with that name exists anywhere, a warning is logged and the workbook is left unchanged.

A cell destination writes to the top-left cell of the reference you supply. If you pass a range, only its top-left cell is used.

On success the formula cell itself is set to an empty string. The response goes only to the destination, never to the formula cell. On failure the formula cell is set to `#VALUE!`.

Execution order

`RW.Tools.AIInsight` runs **after** all data-import formulas, after `RW.Tools.Sort`, and after row suppression. That ordering is deliberate: when you pass a `dataRange`, the model sees the fully expanded, sorted and suppressed dataset rather than the template's placeholder rows.

Examples

An AI summary into a TextBox, with data

```
=RW.Tools.AIInsight("OpenAI", "TextBox1", "Provide a one-paragraph executive summary of this data.", A1:C20, 500)
```

Calls the OpenAI provider, appends the values in `A1:C20` as a Markdown table, and writes the response into the TextBox named `TextBox1`.

A response into a cell

```
=RW.Tools.AIInsight("OpenAI", D5, "Summarize the key trends in this range.", A1:C5)
```

A plain prompt with no data range

```
=RW.Tools.AIInsight("Azure OpenAI", "CommentaryBox", "What are the top three risks in a declining revenue scenario?")
```

A larger token budget

```
=RW.Tools.AIInsight("OpenAI", "SummaryBox", "Summarize this quarter's performance.", B2:F30, 2500)
```

When an argument is wrong

Several of these fail quietly, which is worth knowing before you debug a blank TextBox.

Situation	Result
<code>providerName</code> , <code>destination</code> or <code>prompt</code> is empty or missing	Formula error.
The named provider is not found	Error written to the formula cell; the AI call is skipped.
The provider is found but disabled	Error written to the formula cell; the AI call is skipped.
No TextBox with the <code>destination</code> name exists	Warning logged, workbook unchanged. The formula does not error.
The <code>dataRange</code> worksheet is not found	<code>dataRange</code> is silently ignored and the prompt is sent without any table data.
The provider returns an error	Error written to the formula cell.

Use cases

- **Executive summary on a management pack.** A TextBox on the cover sheet is filled with a one-paragraph summary of the numbers on the sheets behind it.
- **Narrative for a deck.** A TextBox destination pairs well with PowerPoint output, so AI-generated narrative flows into the slide that links to that shape.
- **Per-entity commentary in a burst.** Because the function runs per report, a bursted pack gets commentary written against each recipient's own filtered data rather than the consolidated total.

Notes and limits

- The function is inert in desktop Excel. It only runs inside a Reportworq job.
- Adding it to a workbook has no effect on other formulas or data in that workbook.
- Every call is a real provider call with real latency and cost. A bursted job calls the provider once per report.
- Governance, provider choice, prompt storage and auditing are covered in the AI and Integrations guide.
- Available since around v5.0.0.91. Version availability is approximate.

Related pages

- [Functions overview: which one do I use](#)
- [Connect an AI provider for the prerequisite setup.](#)
- [AI-generated insights in reports](#) for the wider feature this function belongs to.
- [Workday Adaptive functions](#), which are the `AI*` functions that are *not* AI.

Manage scheduled content

how-to

administrator

report author

Scheduled Content is the single place to see and control every schedule in the workspace. It shows a grid of all schedules, a master on/off switch, at-a-glance health cards, and per-row actions to run, edit, enable, or delete a schedule. It is where you monitor whether automated runs are on time and healthy.

When to use it. As your daily check on automated distribution, and whenever you add, pause, run, or retire a schedule. What you see is trimmed by your role, so you view only the schedules and jobs your permissions allow.

Read the schedules grid

Scheduled Content
A unified view of every schedule across the workspace.

Server time: Jul 28, 2026 - 2:52 AM GMT+00:00

Scheduler running all schedules active

Refresh + New schedule

1 Schedule(s)

SCHEDULE	FREQUENCY	NEXT RUN	LAST RUN	REPORTS
☐ Board Packet Delivery Board Packet, Divisional Packet	At 08:00 AM, only on Thursday	N/A	—	2

ⓘ Schedules more than 30 minutes overdue are skipped rather than run, so restarting the server or pausing the scheduler won't trigger catch-up runs.

Open **Scheduled Content** from the side rail. Each row is one schedule, with these columns:

- **Enabled** toggle, **Schedule** (name), **Frequency** (the cron's plain-language description), **Next Run**, **Last Run** (the last actual run), and **Reports**.
- All times are shown in **server time**, noted above the list.

Row actions include **Run now** and **Delete**. Selecting a row opens the [Schedule editor](#).

Turn the master scheduler on or off

The **master scheduler** switch in the top-right header is the global gate for all scheduling. A schedule runs only when the master switch is **on** and that schedule's own **enabled** toggle is on.

1. In the header toolbar, find the labeled switch. On it reads "Scheduler running / all schedules active"; off it reads "Scheduler paused / no schedules will run."
2. To pause everything, turn the switch off and confirm in the **Pause Scheduler** dialog. While paused, the grid also shows an inline note that no schedules will run until you resume.
3. To resume, turn the switch back on.

When to use it. Pause the master scheduler as a single, reversible kill switch during a maintenance window, a source-system upgrade, or a month-end cutover, so no automated run fires against a system that is offline or mid-change. Resume it when you are done. To stop just one schedule, use that schedule's own enabled toggle instead. The master scheduler defaults to **on**.

Only users with scheduler-management permission see the operable switch. Others see a read-only "Scheduler paused" indicator when it is off.

Run, edit, enable, or delete a schedule

- **Run on demand.** Select **Run now** on a schedule's row to run it immediately without waiting for the next occurrence, for example after correcting data that needs to be redistributed now.
- **Edit.** Select the row to open the [Schedule editor](#).
- **Pause one schedule.** Turn its **Enabled** toggle off. It is kept but does not run.
- **Delete.** Use the row's **Delete** action. Deleting prompts a confirmation before it removes the schedule.

Check schedule health

A row of stat cards summarizes health: **Active schedules**, **Next run** (ETA), **Reports scheduled**, and a run-outcome card. When any run failed in the last 24 hours, an attention **Failed (24h)** card shows the count with a **Review failures** link into the [diagnostics](#); when there are none, a neutral last-24-hours card is shown instead.

Use this as a morning operations check: confirm the master switch is on, the next-run ETA looks right, and there is no Failed (24h) count before the business day.

How missed runs are handled (catch-up)

When the scheduler cannot run an occurrence at its due time, because the master scheduler was paused, the host was down, or a prior run was still going, that occurrence becomes **overdue**.

- An occurrence more than **30 minutes** overdue is **skipped**, not run late. This is a fixed rule, not a setting. It stops a flood of stale, backdated runs after downtime, for example an every-15-minutes schedule replaying dozens of times when the scheduler resumes.
- An occurrence within the 30-minute window can still run. It runs through the same pipeline as an on-time run and uses the data **as of the time it actually runs**, not the missed due time.
- Overdue and next-run timing are evaluated in server time. The quiet footer under the grid restates the server time and this catch-up rule.

To force a run that was skipped, use **Run now** on that schedule's row.

Notes and limits

- Scheduled Content manages schedule definitions and health, not the underlying report or distribution design. Edit those in the Job Editor.
- Actions and visibility are constrained by your role, so a limited-role user does not see a complete workspace view here.

Going deeper. To watch scheduled runs as they execute, release a held delivery, or review a run's output, see [Track activity and run history](#).

View and download results

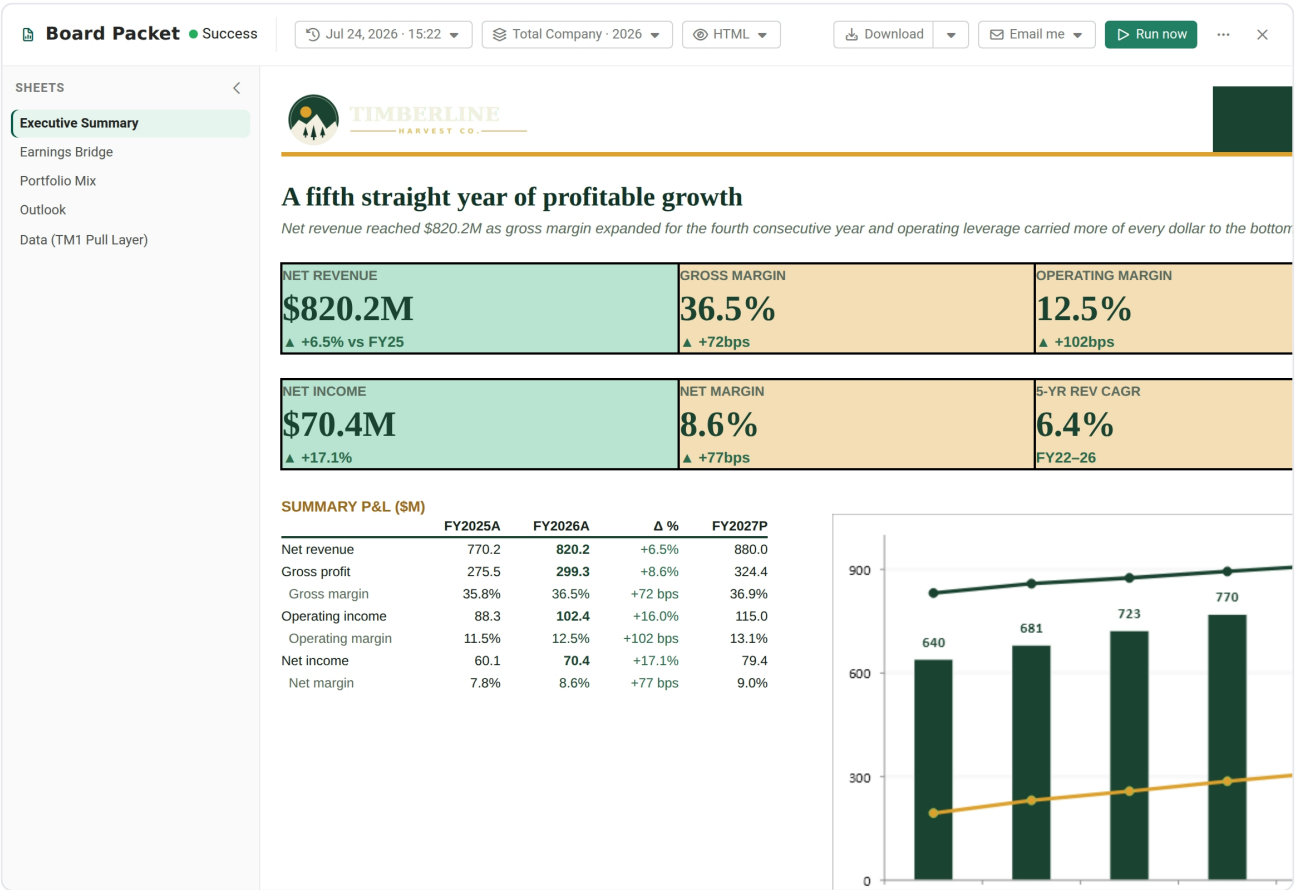
- how-to
- user
- report author
- administrator

The **Report Viewer** shows the rendered output of a report run, without leaving Reportworq and without needing the file. You choose which run to view, which variant (for a burst run), and which format, then read it inline, download the files, or email a copy to yourself.

When to use it. After a run, to read the output on screen, download the deliverables, review a PowerPoint deck slide by slide before it goes out, or send yourself a copy. For Users, this is the main window onto report output.

Open the viewer

1. In the workspace browser, open a report row's ... menu and select **View**. The viewer replaces the browser in place.
2. To return, select the back chevron ("Close, back to the workspace").



Choose the run, variant, and format

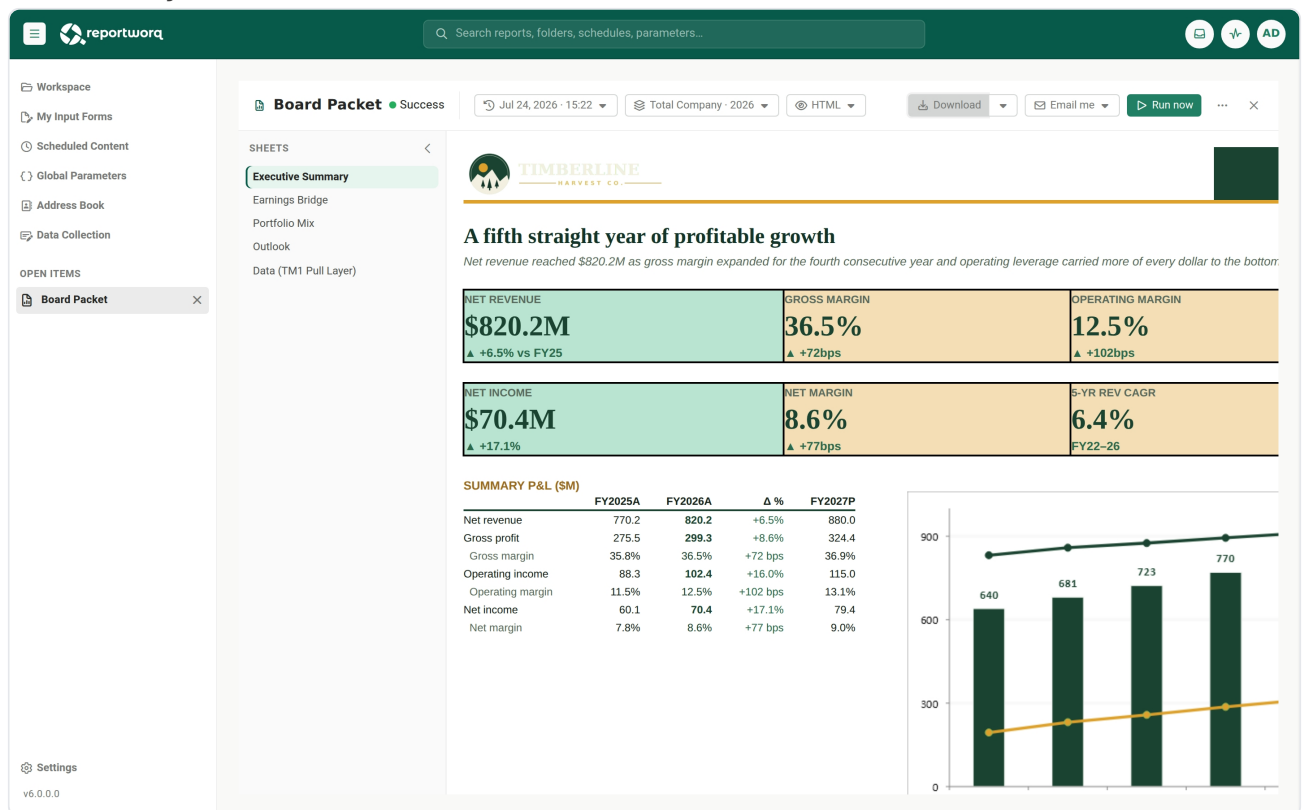
1. **Run picker.** Select which run to view. Runs are listed newest first; each shows its timestamp, a meta line (duration, variations, files, size), and a status chip. A newly

finished run appears here automatically.

2. **Variant picker.** For a burst run with more than one variation, pick the variation (for example a region). A single-variation run has no variant picker.
3. **View as.** For a successful run, choose the format under **View as**. **Markdown** and **HTML** are always offered; **PDF**, **Excel**, **PowerPoint**, and **CSV** appear when the run produced a matching file. The default view is **HTML**.
4. For a **PowerPoint** view, use the **Slides** rail to jump between slides; the selected slide renders full size. The rail scrolls when a deck or workbook has more entries than fit.

Download the output

1. Select **Download** to save the file backing the current view.
2. Open the **Download** menu for more options: each output file (with its size), **Download all in this variant**, and **Download all in this run**. Files are saved individually.



Email a copy to yourself

1. When the run has downloads, select **Email me**.
2. For a single file, it is sent to your signed-in address. For multiple files, choose a per-format item or **Email all formats**.
3. The result is confirmed inline, for example "Sent ... to {your address}."

Refresh the run list

If a run you know has finished is not offered in the run picker, open the ... menu and select **Refresh**. Refresh rebuilds the list from the stored history rather than re-reading what is already in memory, so a completed run whose finish notification went missing appears straight away instead of after a delay.

Notes and limits

- The viewer needs a successful run with output files; only Markdown and HTML are always available, the other formats depend on what the run produced.
- Downloads are per file; there is no zip bundle here.
- AI insights are not a viewer format; they appear in the run's history detail. For the run record, pins, and diagnostics, use [Run History](#).
- A view-only **User** account gets an output-only viewer: view, download, and email, with no edit or run controls.

Going deeper. To see all runs of this report, pin a run, or start diagnostics, see [Track activity and run history](#). If a run failed, see [When a job fails](#).

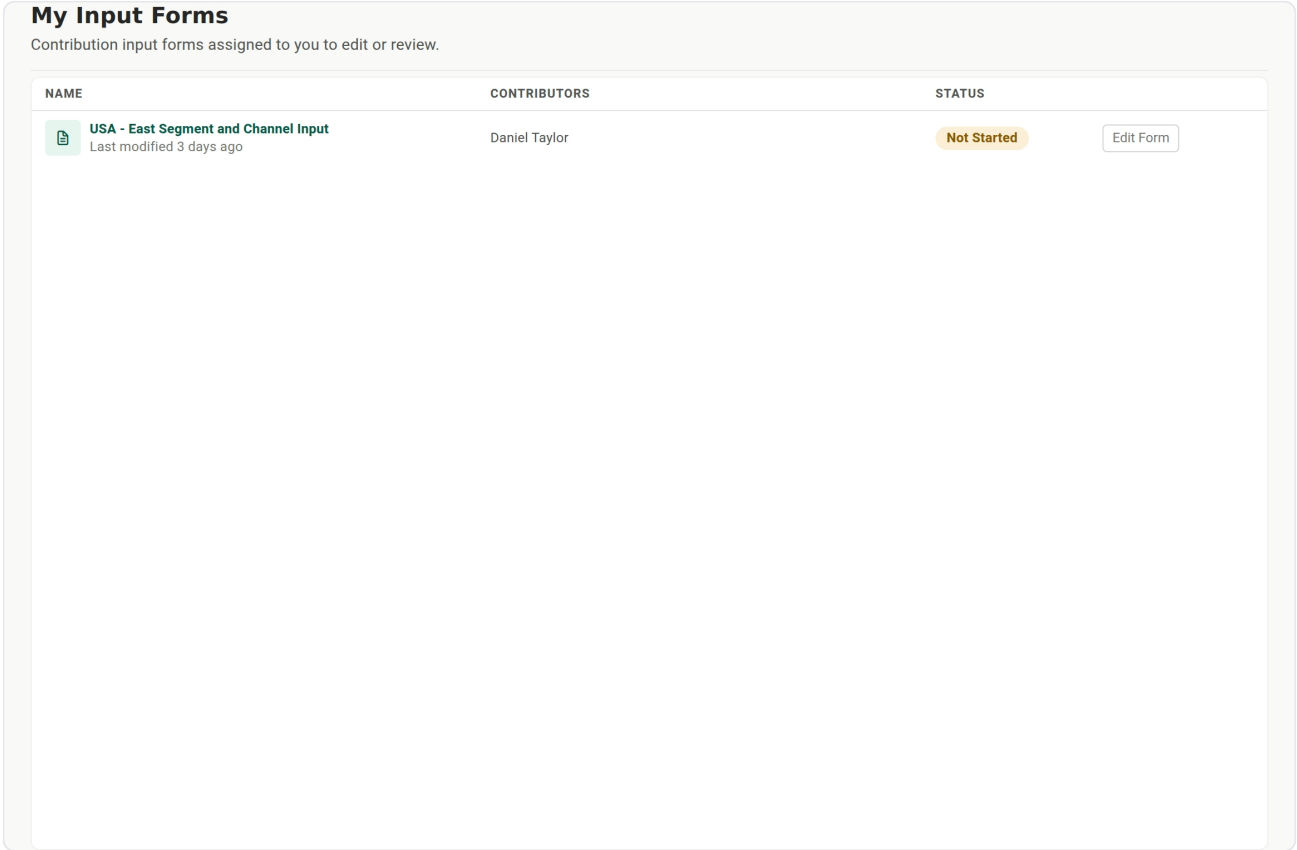
Fill in and submit a form

how-to contributor

When someone adds you to a contribution campaign, your forms appear in **My Input Forms**. Each form is an Excel-based worksheet rendered in your browser: you type into input cells, use the tools on the toolbar above the grid, clear anything the form flags, and submit. This page is the contributor's task guide; if you are building the campaign, see [Author a contribution campaign](#).

Before you start

- You are assigned to a form as a **contributor** or an **approver**. Until then, My Input Forms shows an empty state.
- Open My Input Forms. A form you own opens as **Edit Form**; a form you only review opens as **Review Form**.



A quick tour of the form

An open form has three regions:

- The **planning toolbar** runs across the top, above the grid. It holds the quick tools (Recalculate, Highlight, Gridlines, Undo, Redo, Comment), the **Adjust, Spread**, and **Paste special** operators, a live selection summary, and the workflow actions

(Submit, or Approve and Reject).

- The **grid** fills the middle. Input cells you can edit are tinted; calculated rows are read-only. Sheet tabs sit along the bottom.
- The **Details dock** is a slim rail on the right. It is **collapsed by default**; open it to read the workflow status, the history thread, and a per-cell list of your changes and validation. See [Track your work in the Details dock](#).

USA - East Segment and Channel Input

Not StartedCloseSubmit for approval

AdjustSpreadPaste special1 cellΣ0.049

Segment & Channel Budget Input for USA - East

Enter Volume Δ%, Price/Mix Δ%, and Variable Cost % by line and channel.

•Direct / Field Sales carries the highest contribution protect volume there before shifting mix to Dealers.

•eCommerce variable cost % runs highest across every line (fulfillment + smaller order size) confirm freight assumptions before submitting if you're increasing eCommerce volume.

•Fixed cost plan defaults to FY26 Actual + previously approved headcount adds. Flag any one-time items in the comment field before routing for approval.

Line	Channel	FY26 Net Sales	Volume Δ%	Price/Mix Δ%	Plan Net Sales	Variable Cost %	Contribution \$	CM %
Sawmills & Mill Equipment	Direct / Field Sales	\$17,804,256	4.9%	-0.9%	\$18,508,575	62.0%	\$7,033,258	38.0%
Sawmills & Mill Equipment	Dealers & Distributors	\$16,154,328	50.0%	0.7%	\$24,401,112	65.0%	\$8,540,389	35.0%
Sawmills & Mill Equipment	Retail Partners	\$0	0.0%	0.0%	\$0	66.0%	\$0	0.0%
Sawmills & Mill Equipment	eCommerce	\$0	0.0%	0.0%	\$0	70.0%	\$0	0.0%
Sawmills & Mill Equipment — Subtotal		\$33,958,584			\$42,909,687		\$15,573,648	36.3%
Hardwood Lumber & Millwork	Direct / Field Sales	\$9,335,621	2.7%	0.0%	\$9,587,683	62.0%	\$3,643,319	38.0%
Hardwood Lumber & Millwork	Dealers & Distributors	\$17,200,806	6.4%	-0.9%	\$18,136,943	65.0%	\$6,347,930	35.0%
Hardwood Lumber & Millwork	Retail Partners	\$11,769,761	6.5%	-0.1%	\$12,522,260	66.0%	\$4,257,568	34.0%
Hardwood Lumber & Millwork	eCommerce	\$3,811,512	5.7%	0.0%	\$4,028,768	70.0%	\$1,208,630	30.0%
Hardwood Lumber & Millwork — Subtotal		\$42,117,700			\$44,275,654		\$15,457,448	34.9%
Hand & Power Tools	Direct / Field Sales	\$14,523,019	6.8%	-0.6%	\$15,417,521	62.0%	\$5,858,658	38.0%
Hand & Power Tools	Dealers & Distributors	\$2,225,240	4.9%	0.0%	\$2,334,276	65.0%	\$816,997	35.0%
Hand & Power Tools	Retail Partners	\$3,412,992	4.3%	0.0%	\$3,559,751	66.0%	\$1,210,315	34.0%
Hand & Power Tools	eCommerce	\$1,311,664	7.7%	0.0%	\$1,412,662	70.0%	\$423,799	30.0%
Hand & Power Tools — Subtotal		\$21,472,915			\$22,724,210		\$8,309,769	36.6%
Seed, Grain & Feed	Direct / Field Sales	\$10,333,517	4.6%	0.6%	\$10,873,712	62.0%	\$4,132,010	38.0%
Seed, Grain & Feed	Dealers & Distributors	\$4,208,142	5.9%	0.0%	\$4,456,422	65.0%	\$1,559,748	35.0%
Seed, Grain & Feed	Retail Partners	\$13,530,000	7.9%	0.0%	\$14,598,870	66.0%	\$4,963,616	34.0%
Seed, Grain & Feed	eCommerce	\$8,852,969	5.8%	0.5%	\$9,413,273	70.0%	\$2,823,982	30.0%

Budget EntryFixed CostsSummary

Fill in the form

1. Select a form from the list to open it.
2. Select an input cell and type. Press **Enter** to commit the value, **Escape** to cancel the edit, or **Delete** to clear a cell. Edited cells show a modified marker.
3. Use the form's built-in controls where they appear:
 - A **checkbox** cell toggles a value on or off.
 - A **dropdown** (list) cell offers a set of allowed values.
 - A **filter** control scopes the visible rows. It has a search box; once you apply criteria, **Select All** relabels to **Select All Selected**, and you can pick several criteria together.
4. Navigate large forms with the **row and column groups** (collapse and expand the gutters) and the **frozen panes** that keep headers in view. Select ranges with **Shift+arrow** and **Ctrl+Shift+arrow**, as in Excel. Shift-click or Ctrl-click cells to build a multi-cell selection for the operators below.

Use the planning toolbar

The toolbar above the grid is where you do the planning math. It appears when the form allows editing.

Quick tools

Left to right:

- **Recalculate** re-runs the form's calculations. In manual-calculation mode, this is how you apply buffered edits (see [Notes and limits](#)).
- **Highlight input cells** is a toggle that tints every cell you are allowed to edit, so you can see at a glance where input is expected.
- **Toggle gridlines** shows or hides the grid lines.
- **Undo** and **Redo** step back and forward through your edits.
- **Comment** adds or opens a note on the selected input cell. Select an input cell first; if none is selected, the form prompts "**Select an input cell to comment.**" A cell that carries a comment shows a bookmark you can reopen from the Details dock.

Operators: Adjust, Spread, and Paste special

Select one or more input cells, then open an operator dropdown to apply a bulk change. The **selection summary** next to the operators reads `N cells · Σ total` (and "**no input cells**" when nothing is selected), so you always know what a change will touch.

- **Adjust** changes each selected cell in place: **Set to**, **Add / subtract**, **Multiply**, **Uplift %** (grow or shrink by a percentage), **Round to** (round to the nearest multiple), and **Clear**.
- **Spread** distributes a figure across the selection: **Equal**, **Proportional** (in the same ratios as the current values), **Straight line** (interpolate from a start to an end value), and **Growth %** (a start value, then compound each next cell).
- **Paste special** reworks a copied range as you paste it: **Paste + %**, **Paste + add**, and **Paste × factor**. These need a range on the clipboard, so **Copy first**; otherwise they stay disabled.

Choosing an operator opens a short **param form**: a numeric field (or two) and a **live preview** that shows how many cells will change and the new total (`→ N cells · new Σ ...`) as you type. Select **Apply** (or press **Enter**) to make the change, or **Cancel** (or press **Escape**) to close without changing anything. **Clear** applies immediately with no param form. Each applied operator shows a brief confirmation, for example "**Uplift % applied to 4 cells.**" Operators that the current selection or clipboard cannot satisfy stay disabled.

Clear validation before you submit

Some cells carry validation rules that surface as markers:

- A **blocking** marker (red) means a required rule has not passed, and **Submit stays disabled** until you fix it. The dock's Workflow banner reads "**Validation errors are blocking the submission of this form,**" and the Details tab reads "**Validation errors are preventing the submission of this form.**"
- A **warning** marker is advisory and does not block submission.

The Details **rail badge** turns **orange** and shows the blocker count whenever blockers exist, so you can see there is something to fix without opening the dock. Open the **Details** tab to see each flagged cell, then fix the value each blocker points at until the count clears. The toolbar's Submit button also shows the reason it is disabled, for example "**3 blockers.**"

Copy and paste

Paste works, with rules worth knowing:

- Copy and paste a **single cell or one contiguous rectangle** only. Non-adjacent (multi-block) selections cannot be pasted.
- A paste anchors at the **top-left** of the target and fills down and right.
- **Non-editable cells are skipped.** If your copied rectangle covers a mix of input and locked cells, values land only in the input cells and the rest are silently skipped. There is no error, so check that everything you intended actually landed.
- Cross-application paste is supported: you can copy from Excel or Notepad into the form, and out of it.
- To transform a copied range as you paste it (add a percentage, add an amount, or multiply), use the toolbar's **Paste special** operator instead of a plain paste.

Track your work in the Details dock

The **Details dock** on the right is where you read status and drive the workflow. It is **collapsed by default** to a slim rail labeled **Details** with a count badge; the grid uses the width until you open it. Select the rail to open the dock, and select the collapse control (») in its header to close it again. Reportworq **remembers your choice** the next time you open a form.

The open dock has two tabs.

Workflow tab

- A **status banner** at the top shows the current workflow message. If the form is locked by someone else it reads "**Form Locked. This form is being edited by another user,**" with an **Unlock Form** action; if validation is blocking submission it says so; otherwise it shows the campaign's own message.
- The **System msgs** and **Grouped** chips filter and group the automatic (system) entries in the thread.
- The **history thread** lists the workflow so far, oldest to newest: who acted, the action (Assigned, Submitted, Approved, Rejected), their note, and when.
- A **composer** is pinned at the bottom. Type a note under "**Enter a comment below,**" then use **Submit** (contributor) or **Approve / Reject** (approver). **Recall Form** appears here too when you can recall. These buttons are gated the same way as the toolbar's, and show the reason when disabled.

If you only review a form (no submit, approve, or recall permission), you see the history thread but no composer.

Details tab

The Details tab lists your changes and validation, grouped per sheet as a collapsible "{Sheet} ({count})" header. Each row shows:

- A **command** on the left: undo this change, revert the cell to its submitted value, or (read-only) a formula marker.
- A **validation status** icon: a blocker, a warning, or a passing check.
- A **clickable cell reference** that jumps the grid to that cell, switching sheets if needed.
- A **comment bookmark** if the cell carries a note.
- A **value line**, "Value is 'X', was 'Y'," and "**Last modified by**" the person who changed it.

Settings menu

Select the **gear** (⚙) in the dock header for form settings:

- **Switch to HTML Grid / Switch to Server Grid** changes the render engine (the label reflects the current one).
- **Reset Form** reverts every change back to the starting point (this is destructive and asks you to confirm).
- **Show / Hide System Messages** and **Group / Ungroup System Messages** control the automatic entries in the Workflow thread, matching the **System msgs** and **Grouped** chips.

Submit, and what approvers do

You can act from either the toolbar or the dock composer.

1. When the form is complete and no blockers remain, select **Submit for approval** on the toolbar (or **Submit** in the composer).
2. From the toolbar, a small **comment popup** opens where you can add an optional note before you confirm. The composer sends the note you already typed there.
3. If you need to withdraw a submission, select **Recall Form**.

When the campaign uses approvals, an assigned **approver** opens the submitted form and selects **Approve** or **Reject**; the Workflow banner names who submitted the form and prompts them to review it. **Reject** is styled as a destructive action and, like Submit, opens a comment popup to explain the decision; the form then returns to the contributor to revise and resubmit. Only **one approval** is needed to close a form, no matter how many approvers are assigned.

Comments save only when you submit

This is the trap to remember. **A workflow comment is saved only on a workflow event, that is, on Submit, Approve, Reject, or Recall.** Closing a form saves your *data* work in progress but **not** an unsaved workflow note. If you type a note in the composer and close the form without submitting, the note is lost. Submit (or approve, reject, or recall) to keep it. (Cell comments added with the toolbar's **Comment** tool are attached to their cells and are separate from the workflow note.)

Your data is safer: undo and redo history persists across sessions, and a session that times out on inactivity saves your modifications first ("Your contribution session has automatically ended due to inactivity. All modifications have been saved safely.").

Notes and limits

- A form can be **locked** while someone else edits it. The dock's Workflow banner shows the lock and an **Unlock Form** action, which overrides the lock, so coordinate before you take it.
- **Reset Form** (in the dock's Settings ⚙ menu) reverts every change to the starting point.
- **Export and Import to Excel** (Download and Upload in the header) are available when the campaign allows them. You can export the form, edit it offline, and import the edited workbook back.
- In **manual calculation** mode, your edits are buffered and only recalculated when you ask. The dock shows an amber callout, "**{N} uncommitted edit(s) · Recalculate needed,**" and Submit stays disabled until you press **Recalculate** on

the toolbar.

- A blocking validation marker always prevents submission; a warning never does.

Going deeper. The input cells, checkboxes, dropdowns, filters, and validation you see are defined by contribution formulas in the source workbook. See [Contribution functions](#).

Collect data with an input workbook

how-to report author

Reportworq can collect numbers two ways. A **contribution campaign** generates a form per contributor, tracks who has submitted, and routes approvals. **Data collection** does none of that: it sends a workbook out, lets the recipient fill it in and send it back, and posts the values to the target system when it arrives.

The two are separate processes, and a round uses one or the other. They are not two halves of the same feature, and a campaign does not hand its data to data collection. This page covers the workbook route. For the campaign route, start at [Contribution \(write-back\) overview](#).

How the two differ

	Contribution campaign	Data collection
What the contributor gets	A generated form, opened in Reportworq	A workbook, usually delivered by email
Where they work	In the product, signed in	Wherever they open the workbook, including offline
How input comes back	Submitted from the form	Returned by email to a monitored mailbox, or uploaded
What writes to the plan	The campaign's Export action, which writes to the target directly	The hidden write-back formulas inside the returned workbook
Tracking	Assignment, status per contributor, reminders, approvals	The write-back activity list, one entry per workbook
Where it is watched	The campaign dashboard	Settings, under Data Collection

When to use it

Reach for the workbook route when the ceremony of a campaign is not worth it, or would not work at all.

- **Your contributors are outside the business, or offline.** A regional partner or a franchisee can fill in a workbook without an account and without a connection.
- **The round is small or occasional.** For a handful of submissions once a quarter, assignment, reminders, and approvals are overhead rather than help.

- **The workbook already exists.** If a spreadsheet is already the way a team submits numbers, and it already carries its write-back formulas, data collection keeps that habit and automates the posting.
- **You want the round to live in email.** Send the workbook as a normal distribution, and let replies land in the monitored mailbox.

Choose a campaign instead when you need to know who has and has not submitted, when submissions need approval before they count, or when each contributor must see only their own slice.

Before you start

- The **Contribution** capability must be licensed. Data collection is part of it and has no separate entitlement.
- An administrator must have configured **Data Collection**, and for the email route the monitored mailbox. See [Data collection processing](#).
- You need a connection to the target system, and rights to write to it.

Build the input workbook

1. Build the workbook the way you want the contributor to see it, with the input cells where they would expect them.
2. In cells the contributor does not need to see, usually a hidden row or a hidden worksheet, add a write-back formula for each value you want posted. The formula names the target and the intersection, and points at the cell the contributor types into.
3. Match the function to the measure and the target system. Planning Analytics uses `DBS` and `DBSW` for numeric measures and `DBSS` for strings; SQL, Workday Adaptive, and Anaplan each have their own. For signatures, arguments, and the traps, see [Contribution functions](#).
4. Protect or lock everything that is not an input cell, so a contributor cannot move the values out from under the formulas.

The write-back formulas do nothing until the workbook is processed. They stay inert while the contributor works, which is why the workbook is safe to open in Excel on any machine. They evaluate only when the data collection engine runs the returned file in write-back mode.

Send it out and get it back

1. Distribute the workbook as you would any other report, and burst it if each recipient should get their own copy. See [Distribution overview](#).
2. The contributor fills it in and returns it, either by replying to the monitored mailbox or by having someone upload it.
3. An administrator processes the item, or it processes on arrival if **Automatically process uploaded files** is turned on.
4. Check the outcome in the write-back activity list. Each item shows its file name, who submitted it, when it arrived, when it was processed, and its status.

Notes and limits

- **A returned workbook is trusted.** Whatever the write-back formulas resolve to is what gets posted, so anyone who can edit the workbook and get it into the mailbox can write to the plan. Use a campaign when you need per-contributor control over what can be changed.
- **There is no submission tracking.** Nothing tells you that eleven of your fourteen regions have replied. If that matters, use a campaign.
- **A consolidated cell is rejected by default** on a numeric write, matching the Planning Analytics Excel add-in. Turning on **Proportional Writeback** posts the value for allocation across the leaf cells instead. String writes are unaffected.
- **Removing provider formulas removes the ability to write back**, unless you keep the input formulas. When you distribute a workbook that is meant to come back, check the **Formula Options** setting on the output. See [Output formats](#).

Related topics

- [Contribution functions](#) for the write-back formulas in depth
- [Contribution \(write-back\) overview](#) for the campaign route
- [Data collection processing](#) for the engine, the mailbox, and troubleshooting

Store files in the workspace

how-to

report author

administrator

A workspace can hold uploaded **files** - Excel workbooks, Word and PowerPoint documents, PDFs, images, archives, and text - right beside its reports, campaigns, and data models. A file is a first-class workspace item: it lives in the folder tree and you **move, rename, copy, delete, and secure** it exactly like any other item, with the same access control.

This is the governed home for the supporting files your reports depend on - a logo referenced by every board pack, a source workbook a report reads from, a PowerPoint template - so they are versioned by upload, secured by the workspace, and picked by name rather than tracked outside the product.

Before you start

- File storage must be turned on for the server. It is **on by default**; an administrator can disable it, or restrict which file types are allowed, under **Settings > Configuration** ("Allow storing files in workspaces"). If it is off, you will not see the Upload button or the Files filter.
- Uploading requires authoring permission. A view-only user can download files they are allowed to see, but cannot upload or change them.
- Each file can be up to **500 MB**, and its extension must be on the allowed list.

Upload a file

1. Browse to the folder where the file should live.
2. Choose **Upload** in the browser toolbar and pick one or more files. They are created in the current folder.

If you upload a file whose name matches one already in that folder, Reportworq asks with an **Overwrite File** prompt:

- **Confirm** replaces the file's contents **in place**, keeping the same item. Any report or campaign that links to it keeps working.
- **Cancel** skips that file and lists it as "not replaced." The existing file is never renamed or duplicated.

Drag and drop, and zip archives

You can also drag files from your computer straight onto the browser; they land in the current folder. Dropping a **.zip** gives you a choice:

- **Extract into folders** recreates the archive's folder structure as workspace folders, with each allowed file as an item.
- **Store as one file** keeps the zip itself as a single file item.

Entries with disallowed extensions, and any unsafe path entries, are skipped and reported. A zip that is actually a workspace **export** bundle is imported instead of stored.

Download a file

Each file row has a **Download** action that streams the stored file to your browser.

Find files quickly

When file storage is on, the browser's filter bar gains a **Files** chip with a count, so you can narrow the list to files alone.

Use a file as a report source

The Job Editor's report picker has a **Workspace** route backed by these files. Upload a source workbook once, then pick it there by folder and name - the same file library, the same security, no separate upload per report. See [Choose the source workbook and worksheets](#).

Security

Files inherit the workspace's overall access policy and can carry their own per-item access list, just like reports. There is no separate file-permission model. See [Item properties](#) and [Roles and what you can do](#).

Notes and limits

- Moving or renaming a file never rewrites its contents; a copy duplicates it; a delete is a soft delete.
- Uploads are capped at 500 MB per file and limited to the allowed extension list an administrator configures.

Parameters reference

AUTH-02 reference report author

This page is the exhaustive reference for parameters on the **Parameters** step of the Job Editor: every parameter **type**, how each parameter is **mapped** to a worksheet, how list-valued parameters **replicate** output and **cross-join**, and how **parameter sets** work. For the task walkthrough, see [Parameterize a report](#). For variables, cascading, and repeaters, see [Advanced parameters](#).

Bursting, replication, and burst sets: the words first

These three words get used interchangeably and they are not the same thing. Getting them straight makes the rest of this page, and the Bursting step itself, much easier to read.

Bursting is the outcome: taking one report and producing many versions of it from one set of parameter values. What gets multiplied can be a single worksheet, a group of worksheets copied together, or an entire report output.

There are three ways to achieve it, in increasing order of control:

Technique	What it is	Reach for it when
Replication	A parameter that holds more than one value fans the output out, one copy per value. Set the parameter's Replication mode and you are done. No Bursting step involved.	The everyday case. A cost-center query returns 40 cost centers and you want 40 versions.
Parameter sets	An explicit, typed list of parameter <i>combinations</i> , one output per line.	The combinations are sparse. See Parameter sets below.
Burst sets	The Bursting step, unlocked by flagging a parameter Burstable . Gives per-file control: you can specify the parameter values, or a query, for each generated file individually.	You need fine-grained control over each file, or per-recipient delivery.

The distinction that trips people up: you do **not** have to flag a parameter **Burstable** to get many outputs. If a parameter already has multiple values, from a Planning Analytics subset, an MDX query, a SQL query, or a typed Text List, and you set a replication mode, **it is already fanning the output out**. That is replication, and it needs nothing from the Bursting step.

Flag a parameter **Burstable** only when you actually want the **Bursting** step, because you need burst sets, per-file parameter control, or recipient-driven delivery. Throughout

this guide, output fanned out by a parameter's own values is called **replication**; **burst set** means specifically the Bursting-step feature.

See [What bursting is and when you need it](#).

Parameter types

The **Type** dropdown is plugin-driven: the five **Native** types are always present, and each connector contributes its own types when a connection of that kind exists. The default type is **None**.

Type	Source	What it holds
None	Native	An untyped, free parameter.
Text Value	Native	A single literal value.
Text List	Native	Allowed values entered one per line. The only type that supports parameter sets and bursting import and export.
Excel Range	Native	Values read from an Excel range.
Global Parameter	Native	References a workspace or global parameter's value.
Planning Analytics Subset	TM1 / Planning Analytics	Values from a TM1 or Planning Analytics subset.
Planning Analytics Dimension	TM1 / Planning Analytics	Elements of a Planning Analytics dimension or hierarchy.
Planning Analytics MDX Query	TM1 / Planning Analytics	Elements returned by an MDX query. Supports <code>%param:Name%</code> and <code>%global:Name%</code> runtime tokens.
Planning Analytics Cube Value	TM1 / Planning Analytics	A value read from a cube cell or tuple. Dimension selections support <code>%param:Name%</code> and <code>%global:Name%</code> tokens.
SQL Query	SQL	Values from a SQL query. Supports <code>%param:Name%</code> and <code>%global:Name%</code> runtime tokens.
Workday Adaptive Dimension	Workday Adaptive	Adaptive dimension members (specification-based).
Oracle Smart View Dimension	Oracle EPM Cloud / Smart View	Smart View point-of-view dimension (specification-based).
Anaplan Dimension	Anaplan	Anaplan dimension members (specification-based).
Vena Dimension	Vena	Vena dimension members.

Connector types appear only when a connection of the matching kind exists: a Planning Analytics connection for the Planning Analytics types, a SQL connection for SQL Query, and so on.

Per-worksheet mapping (Location)

Parameter definitions are held at the **report level**, where each name is unique. Each parameter is then **mapped per worksheet** on the Reports step, and its **Location**

determines where its value is read. **A parameter's value is set once, at the report level, not per worksheet.** You can map the same parameter onto many worksheets, and often will, but every one of those worksheets receives the *same* value. There is no way to give one worksheet a different value for the same parameter name. If two worksheets genuinely need different values, they need two parameters.

Location shape	Example	Behavior
Cell or named range	B3 , or a named range	The value is written into the cell, so downstream formulas can read it. Prefer a named range: a cell reference breaks silently if someone inserts a row in the source workbook.
Pivot-table field	PivotTableName.FieldName	The value is applied as a pivot filter (a page field sets the current page item; a row or column field adds an equals filter), not written to a cell.

Replication modes

Every parameter has a **Replication mode** that decides whether the parameter fans the output out, and how. You set it in the parameter's edit panel, under **Parameter Options**. The mode is one of:

Mode	Effect
No replication	The parameter supplies a single value. Nothing is replicated.
One page per item	The worksheet is copied once per value, producing one page per item within the output.
One report per item	A separate output file is produced per value, which integrates with file-naming variables so each file can be named after its value.

No replication

Choose **No replication** when the parameter is a single value that should never fan the output out (a reporting period, one legal entity, a scenario). How you set it depends on the parameter type:

- A **Text Value** parameter is always a single value, so the replication area is hidden and the mode is fixed to **No replication**. The panel shows the note "A text value is a single value, so no replication applies," and there is nothing to choose.

Edit Period



Parameter Type

Text Value



▼ Text Value

Test...

Value

▼ Parameter Options

A text value is a single value, so no replication applies.

☐ Enable this parameter to be used in burst sets

▼ Security

☐ Secured parameter

Controls who can see each output by its value of this parameter.

Only a "One report per item" parameter can be secured - each value then produces its own report, which is the unit access is granted on. "One page per item" keeps every value in one workbook, so there is nothing to gate.

 Save & Close

- For **any other type** (a Text List, a SQL or MDX query, a dimension selection), **No replication** is one of the choices in the mode list. Because a dynamic query can return zero, one, or several values, you also declare what Reportworq should do if the parameter does **not** resolve to exactly one value:

If the parameter does not resolve to one value	What happens
Fail the job (a single value is required)	The run fails with a clear error that names the parameter and the number of values it resolved to, whenever that number is not exactly one (zero, or several). This is the default . Choose it when exactly one value should always come back, so anything else is a mistake worth surfacing rather than papering over.
Use the first value, and omit the pages if there are none	If the query returns several values, Reportworq uses the first one. If it returns none , the worksheets that map this parameter are omitted rather than produced empty, and the whole report is dropped if that leaves it with no worksheets.

Edit Period



Parameter Type

Text List



▼ Text List

Test...

Items

2026-May

▼ Parameter Options

When a parameter contains multiple values you can decide to have Reportworq create multiple pages in a single report or multiple reports for each item. Choose "No replication" when the parameter is a single value that should never fan the output out.

- ☐ One report per item
- ☐ One page per item
- ☒ No replication

This parameter should resolve to exactly one value. If a query returns none or several, choose what to do:

- ☒ Fail the job (a single value is required)
- ☐ Use the first value, and omit the pages if there are none
- ☐ Group & collate pages that use this parameter
- ☐ Enable this parameter to be used in burst sets

▼ Security

- ☐ Secured parameter

Controls who can see each output by its value of this parameter.

Only a "One report per item" parameter can be secured - each value then produces its own report which is the unit access is granted on. "One page per item" keeps

 Save & Close

The single-value check runs when the job builds its run, **before** any worksheets are produced, so a **Fail** choice stops the run early rather than partway through.

Related controls:

- **Collate worksheets** orders "one page per item" replications by parameter across worksheets (for example Income Statement then Balance Sheet for US, then the same pair for UK, then Canada), rather than grouping all copies of one sheet together. See [Job options](#).
- **Burstable** flags a parameter for bursting, which reveals the otherwise-disabled Bursting step and uses the same replication machinery to fan output out per recipient.

Multi-parameter cross-join (Cartesian)

When **more than one** replicating parameter is present, the runtime takes the **Cartesian product** of their value lists. For example, 3 business units cross 4 departments produces **12** worksheets or reports.

Only **One page per item** and **One report per item** parameters take part in the cross-join. A **No replication** parameter is a single value, not a replication axis, so it contributes its one value to every combination without multiplying the count.

Parameter order controls the sequence

The order of the parameters decides the order of the output, because the **first** parameter is the outer loop. With 3 business units and 4 departments:

Business Unit first, then Department, gives you BU1 with departments A, B, C, D; then BU2 with A, B, C, D; then BU3 with A, B, C, D. The output is grouped by business unit.

Order	1	2	3	4	5	6	...
Output	BU1 / A	BU1 / B	BU1 / C	BU1 / D	BU2 / A	BU2 / B	...

Flip the parameters so Department is first, and you get department A with BU1, BU2, BU3; then department B with BU1, BU2, BU3, and so on. Same 12 outputs, grouped by department instead.

Order	1	2	3	4	5	6	...
Output	A / BU1	A / BU2	A / BU3	B / BU1	B / BU2	B / BU3	...

Choose the order that matches how the reader wants to read the pack.

When you do not want every combination

A cross-join is exhaustive, and often you do not need every cell of it. Not every department exists in every business unit. You have two options, and they have very different performance profiles.

Page suppression discards the pages you do not want after they are produced. Set it up on the Reports step. It is the simpler route and it is the right one when most of the combinations are wanted and you are trimming a few. Be aware of what it costs: **the data for a suppressed page is still fetched and the page still generated, then thrown away.** With 12 pages and 2 suppressed, nobody notices. With 5,000 combinations of which you keep 200, you have paid for 5,000.

Parameter sets avoid producing the unwanted combinations at all, because you state the combinations you want rather than deriving them. When the product is large and the retained set is small, this is much the faster route. See below.

Parameter sets

A **parameter set** is a finite, explicit list of combinations, not a cross-join. It applies to **Text-List parameters on cell-based reports only.**

When a parameter set is worth it

This is an advanced feature and it is not the everyday way to fan output out. Most reports want a replicating parameter, which is simpler and needs no combination list. But parameter sets earn their keep in one specific situation, and there they are the only good answer.

Reach for a parameter set when the combinations you want are sparse.

Continuing the example above: a cross-join of business units and departments produces every pairing, but suppose only 30 of those 5,000 pairings actually exist in your organization. A cross-join would query and build all 5,000 and let you discard 4,970 by page suppression, which works and is slow. A parameter set lets you list the 30 real combinations, and only those 30 are produced.

So the decision is:

- **Combinations are dense** (most pairings are real, you want nearly all of them): use replicating parameters and let the cross-join do the work. Trim the few strays with page suppression.
- **Combinations are sparse** (a small, known set out of a large possible space): use a parameter set.

- **You need per-file control** over each output, or recipient-driven delivery: use a burst set on the Bursting step.

The other case for a parameter set is when no query can express the combinations you need. If the pairing of values is a business rule rather than something derivable from the data, listing it explicitly is clearer than contorting a query into producing it.

- Enter each combination as a **pipe-delimited value line**, and set a **pipe-delimited Location**. For example, a value line of `East|Q1` with a Location of `B3|B4` writes `East` into `B3` and `Q1` into `B4`.
- **Each value line produces exactly one worksheet**. Multiple lines fan out one worksheet per line.
- A **part-count mismatch** between the Location and the value (for example a 3-part Location with a 2-part value) is a **silent no-op**: nothing is written and the cells are left blank. No error is raised, so watch for this when debugging an unexpectedly empty output.

Limits

- Parameter names must be **unique within the report**.
- Bursting is available only once a parameter is flagged **Burstable**.
- **Parameter sets** and **cell-writing mappings** work only for cell-based reports. Specification-based providers take the value directly and do not use a cell reference.
- Only **Text List** supports parameter sets, and the reason is structural: a set is a list of literal, pipe-delimited combination lines that you type. A query-driven type returns one value per row, so it cannot express a combination.

Related topics

- [Parameterize a report](#)
- [Advanced parameters](#)
- [Worksheet output options](#)
- [Job options](#)

Assign recipients to bursts

[how-to](#)[report author](#)

A job that fans out into many outputs often needs each output to reach a different person. This page covers the ways an output gets a recipient and how they interact.

A recipient is not required. An output with no recipient is still produced and still delivered, using the job's own destinations: a fixed network folder, a static email list, a SharePoint location. Plenty of jobs never assign one.

The address book is not the only place recipient information can live, and often it is not the best one. If your source system already records who owns each cost center, or the report itself looks the address up on a worksheet, that information is already where it belongs. Use the address book when you need somewhere to keep distribution metadata that has no better home.

The ways an output gets a recipient

Approach	Where the recipient comes from	Best when
Contact mapping	A value resolved at run time, usually the burst or replicated parameter, matched against a field on your contacts.	The source system knows the value but not the delivery details, so the contact record supplies the destinations.
A worksheet value	A cell in the report, for example the result of a lookup on a worksheet, used as the value that contact mapping matches on.	The workbook already resolves the owner, so no second list is needed anywhere.
A Contact column in the burst source	A Contact column in the Excel workbook or the SQL result that generates the burst sets.	The list of outputs and the list of recipients are maintained together.
A contact assigned by hand	The Contact (optional) column on a static burst set.	A short, curated roster where the pairing is stable.
None	The job's own destinations deliver every output.	Everything lands in one folder or goes to one list.

What the address book is for

A contact is a bundle of **distribution metadata**: an email address, a network folder, a SharePoint folder, a Slack channel, a Teams target, a Google Drive location, plus any

custom fields you define. It is not a security boundary and it does not grant anyone access to anything.

The useful part is that once an output is associated with a contact, all of that metadata becomes available on the **Distribution** step. Destination fields and file names can resolve **Contact** tokens against it, so one job can write each output to its owner's folder and name the file after them. An output becomes associated with a contact in one of two ways: you pick the contact by hand on a burst set, or contact mapping resolves one at run time.

That association is worth setting up when you need per-recipient destinations. If you do not, skip it.

Assign a contact on a burst set

Every burst grid has a **Contact (optional)** column, and how you set it depends on the source:

- **Static sets** give you an editable **Choose a Contact** combobox on each row, populated from the address book. Select a contact per set, or clear it to leave the set unassigned.
- **Excel Source** and **SQL Source** resolve the contact from the source itself: a **Contact** column in the workbook, or a matching column in the SQL result. The grid shows the resolved contact name as read-only, or blank if the value matched no contact.
- **Import and export** carry the contact by name, so `burstsets.xlsx` writes the contact name and import resolves it back against the address book.

Assignment is per set and always optional.

Let contact mapping resolve the recipient

Contact mapping resolves a contact at run time by matching a value against a field on your contacts. It is not limited to burst sets: it fills the recipient for **any** output that does not already have one, so it works equally well for a job that fans out by **replication**.

1. Go to the **Distribution** step and open the advanced options (the **Job Options** sidebar).
2. Turn on **Associate Contact with a mapping**.
3. Choose a **Mapping Field**, the contact attribute to match on. The list offers the contact's **Name**, any custom field you have defined on your contacts, and each enabled distributor's destination fields.
4. Build the **Mapping Value**, the expression that resolves to the value to match. It is usually the parameter that the job fans out on. It can also read a value from a cell

in the report, which is how a lookup on a worksheet can drive the routing.

At run time Reportworq resolves the Mapping Value, finds the contact whose Mapping Field equals it, and uses that contact as the recipient.

Field lookups are case-insensitive. The Mapping Value is matched against the contact field with both sides lowercased, so `NORTH`, `North`, and `north` all resolve to the same contact. Casing differences never cause a miss.

Which one wins

When both are in play, precedence is settled:

- **A contact assigned on the burst set wins.** Contact mapping only fills an output whose contact is empty, so it is strictly a fallback and never overrides an assigned contact.
- If the Mapping Value matches **several** contacts, the **first** is used.
- If it matches **none**, the output keeps no recipient. It is still produced, any **Contact** token in the file name or destination fields resolves empty for that output, and delivery falls to the job's own destinations.

Once a contact is resolved, its attributes feed the destination and Contact tokens in the file name resolve against it. See [Name and organize output files](#).

Use cases

- **Owner routing from a custom field.** A job fans out on **Cost Center**, contacts carry a custom **Cost Center** field, and mapping pairs each statement with its owner without a single hand assignment.
- **Recipients maintained with the list.** An entity master workbook pairs each entity with a contact in a **Contact** column, so Excel Source produces both the roster and the routing in one place.
- **A curated roster.** Twelve static sets, each with its recipient picked by hand, because the pairing is stable and there is no system of record for it.
- **Per-recipient folders instead of email.** Each contact's **Network Folder** attribute directs that recipient's output to their own file share, with no email involved.
- **One recipient does not fit.** Most sets take the mapped owner, and one set overrides it with a named contact because that report goes to a controller instead.

Notes and limits

- **Contacts are distribution metadata, not security.** A contact records where and to whom to deliver. It grants no access and gates nothing.
- **For mapping to route everyone,** every value the Mapping Value can resolve to needs a matching contact, or that output has no recipient.
- **The Contact variable type cannot be used inside the Mapping Value,** which guards against a circular reference.
- **If a newly added custom contact field does not appear** in the **Mapping Field** list, refresh the browser to reload the field set.

Going deeper. A resolved contact's attributes drive where and how each output is delivered, and Contact tokens personalize file names and folder paths. See [Destinations](#), [Name and organize output files](#), and [Address book](#).

Name and organize output files

DIST-01

how-to

report author

The **file name** field on the Distribution step builds the name of every generated file from **literal text** plus **variable tokens** (a parameter value, a contact attribute, a global variable, or a timestamp). Because tokens resolve per burst iteration and per recipient, one job produces distinctly named files across parameters, recipients, and runs, and the same tokens work in a destination's folder or path field to organize where those files land.

The worked example is a **monthly P&L pack** bursted per store.

Before you begin

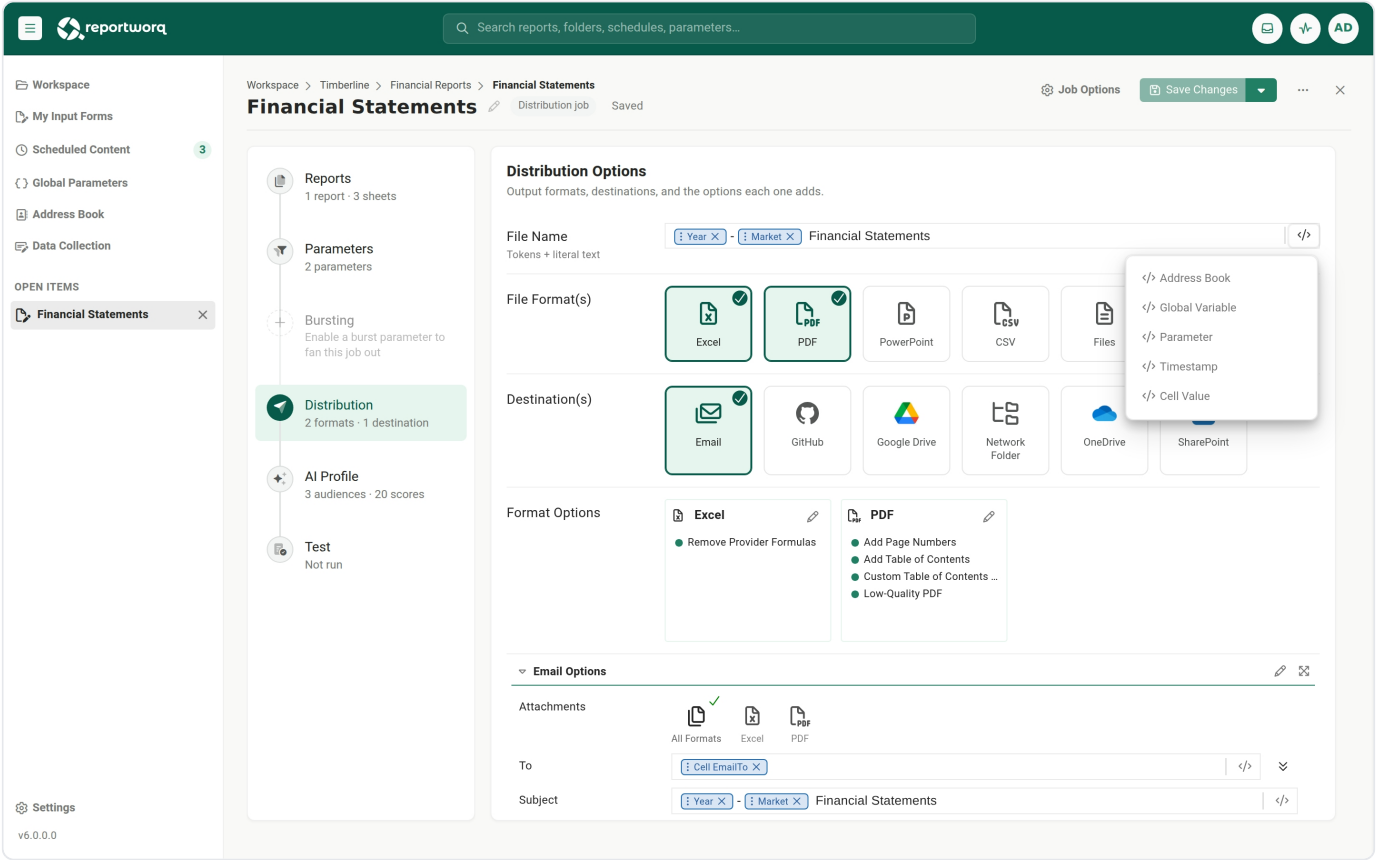
- You are on the **Distribution** step of the Job Editor, with at least one output format selected. See [Output formats](#).
- For **parameter** tokens, the parameters must already be defined on the Parameters step. For **contact** tokens, the job must burst to contacts or use contact mapping. For **global variable** tokens, the global variables must be configured.

Build a file name from tokens

1. Select the **file name** field. It is a pill editor: you type literal text and insert variable tokens between the text.
2. Type the fixed part of the name, for example `P&L_`.
3. Insert a **variable pill** where you want a value to appear, and pick its type:
 - **Parameter**, resolves to the parameter's value for the current burst iteration (for example the store code).
 - **Address Book**, resolves to an attribute of the recipient contact (name, email, and so on).
 - **Global Variable**, resolves to a configured global value.
 - **Timestamp**, resolves to the run time formatted with a **Timestamp Format** code you pick (for example `g` for short date and time).
4. Configure the pill in its popover and choose **Save and Close**. The placed pill shows the field it will resolve.
5. Add more literal text and pills as needed. A name like `P&L_[Store]_[Timestamp]` produces `P&L_Store12_...`, `P&L_Store13_...`, each with the run date embedded.

The format-specific extension (`.xlsx`, `.pdf`, `.pptx`, `.csv`) is appended automatically per output format; you set the **base name** only.

Before you insert a parameter pill, check that parameter's **replication mode**. Only a parameter set to **No replication** or **One report per item** belongs in a file name, see the note in [Notes and limits](#).



Contact token display options

The contact token is listed as **Address Book** in the insert-variable menu. When you insert one, pick which recipient attribute it renders:

Contact display option	Resolves to
Name	The contact's name.
Email Address	The contact's email address.
Network Folder	The contact's network-folder destination value.
Google Drive Email / Id / Folder	The contact's Google Drive email, id, or folder.
SharePoint Folder	The contact's SharePoint folder.
Slack Channel	The contact's Slack channel.
Teams	The contact's Teams destination.

Organize output into folders automatically

The same tokens work in a **destination's folder or path field** (Network Folder, SharePoint, Box, OneDrive, IBM Planning Analytics application folder). This is the key organizing trick:

Putting a variable in the path auto-creates the folder tree at delivery time.

When a folder-based destination path contains a token, Reportworq resolves it per iteration and **creates any missing folder segments** as it delivers. You do not pre-create the folders.

When to use it. You are bursting one report per store (or per entity, region, or cost center) and want each store's file in its own folder. Put the **store parameter** token into the Network Folder or SharePoint path, for example `\\FileServer\Reports\[Store]\`, and the run creates `...\Store12\`, `...\Store13\`, and so on, dropping each store's file where it belongs.

The same works for dated archives: a **Timestamp** token whose format yields dated segments builds year and month folders, for example a format like `MMMM d yyyy` creates a folder named "August 12, 2025", so successive runs archive cleanly instead of overwriting each other.

Use cases

- **Per-store burst.** `P&L_[Store]_[Timestamp]` names each store's file uniquely and dates it.
- **Recipient-personalized forecast.** A **Contact, Name** token yields `Forecast_JaneDoe.pdf` per recipient when the job bursts to contacts.
- **One folder per entity.** A parameter token in the SharePoint path drops each entity's pack into its own library folder, created on delivery.
- **Dated archive.** A Timestamp token in both the name and the network-folder path keeps every run side by side instead of overwriting the last one.

Notes and limits

- **The file name is a base name only.** The extension is fixed per output format.
- **Only use a parameter pill for a parameter that produces its own file.** Check the parameter's **replication mode** on the Parameters step first:

- **No replication** and **One report per item** are safe. Each resolves to a single value for the file being written, so the name comes out clean and distinct.
- **One page per item** is not. That mode keeps every value in **one** workbook, one page per value, so there is no separate file per value to name. The token resolves to all of the values joined by commas, which produces an unwieldy name like `P&L_East, Central, West.xlsx` rather than three files.
- **File names come out the same on every host.** Reportworq replaces each character that cannot safely appear in a file name with a single underscore, using one fixed set: `" < > | : * ? \ /` plus control characters. The set does not depend on the operating system, so the same job produces the same file name whether the server runs on Windows or Linux. A token that resolves to `UK|Sales` delivers `UK_Sales`, and a colon-bearing timestamp such as `-05:00` delivers `-05_00`.
- **Reserved names and trailing punctuation are not adjusted.** Sanitization covers characters, not Windows reserved device names (`CON`, `NUL`, and similar) or a name ending in a dot or a space. If a token could resolve to one of those on its own, add literal text around it so the name is never just the reserved word.
- **A token that references an unset value may resolve empty**, for example a Contact attribute when the job does not burst to contacts.
- **Re-runs.** A file name with no Timestamp token resolves to the same name every run, so each run overwrites the last. In a network folder or SharePoint destination, add a **Timestamp** token when you want to keep the history. Cloud destinations such as Box and OneDrive version or replace a same-named file in place rather than duplicating it.

If overwriting **is** the intent, for example a single live dashboard file that always sits at the same path, leave the timestamp out. Be aware that the overwrite can fail when someone has the target file open: SharePoint and Windows file shares lock a file that is open for editing, so the delivery to that destination errors and the file keeps its previous contents. Other destinations in the same run are unaffected, and the run's status reflects the error. If a file is meant to be refreshed in place on a schedule, tell its readers to close it before the run window, or deliver to a timestamped name instead.

Upgrading from an earlier release on a Linux host? File names may change. On Linux (which includes container and Azure App Service deployments), earlier releases replaced only `/`, so `\ : * ? " < > |` passed straight through into the delivered file name. Those characters are now replaced with an underscore each, matching what Windows always did. A file name built from a colon-bearing timestamp format, or from a token that resolves to something like a `server:cube:element` reference, therefore arrives under a slightly different name than it did before. Windows deployments are unaffected. If anything downstream matches delivered file names exactly, a mailbox rule, a load script, or a watched folder, check its patterns after the upgrade.

Going deeper. The message body (email, Slack, Teams) accepts a *superset* of these tokens, including live cell values, charts, images, and screenshots pulled from the workbook. See [Email messages and templates](#).

Excel: charts and maps

reference

report author

New chart on the companion pane's **Home** hub opens the Reportworq chart and map builder. You choose a visual, bind it to ranges on the worksheet, watch a live preview, and then place it on the sheet.

What lands on the sheet is a **picture that carries its own definition**. The definition travels with the picture, so Reportworq can redraw the visual later from whatever the bound cells hold at that moment, either on demand in Excel or automatically when the job runs. There is no separate stored copy to keep in step.

For how this compares with a native Excel chart, and when to use which, see [Charts and maps](#).

Before you start

- The shared prerequisites in [The Office add-ins](#): HTTPS on the server, the add-in installed, and a signed-in licensed account.
- **Data already on the worksheet**, in the shape the chart type expects. See [Chart types](#). You can lay it out by hand, import it as a report first, or use **Sample Data** to write an example block.
- An **AI model configured**, only if you want **Create with AI** to work out the bindings for you. Without one the builder falls back to a fixed rule of its own, and you can always bind the ranges yourself.

Build a chart

1. On **Home**, select **New chart**.
2. **Choose a type**. **Column**, **Line**, and **Bar** are tiles. **More...** opens the rest: Stacked column, Stacked bar, Pie, Waterfall, Combo, Sankey, Range bars, Range area, Heatmap, and Map.
3. **Name it**. **Name** identifies the chart in the workbook hub and in the sheet. **Title** is the heading drawn on the visual itself, and it takes either typed text or a cell reference, so the heading can come from the data.
4. **Bind the data**. Enter each range, or use the picker beside a field to select cells on the sheet. What you bind depends on the type. See [Chart types](#).
5. **Check the preview**. The **Preview** dock at the foot of the pane redraws as you bind. **Refresh** rebuilds it.
6. Select **Import chart**.

The visual is placed on the sheet as a picture and joins the **Charts & maps** list on **Home**.

Edit, refresh, or delete a chart

The **Charts & maps** list on **Home** holds **every** Reportworq chart on the workbook, across all worksheets, each labeled with its type. Each row offers three actions:

- **Edit chart** reopens the builder, seeded from the picture's own definition. In edit mode the header reads **Edit chart** and the action button becomes **Update chart**, which redraws the picture in place and leaves you in the builder.
- **Refresh chart** redraws that one picture from the current worksheet values.
- **Delete chart** removes the picture from the workbook.

The workbook-level **Refresh** under **On this workbook** spans reports and charts together.

Selecting a chart's picture on the sheet highlights its row in the hub, which is the quickest way to find the definition behind a visual you are looking at.

Chart types

Most types are **categorical**: a range of labels plus one or more ranges of values. Three of them read a differently shaped block instead, the heatmap, the Sankey, and the map.

Type	What it draws	What you bind
Column	Vertical bars, one group per category	Category (labels) range plus a Values range per series
Bar	The same bars drawn horizontally	Category range plus a values range per series
Line	A trend across the categories	Category range plus a values range per series
Stacked column	Series stacked within each category, vertically. Positive and negative values are both handled	Category range plus a values range per series
Stacked bar	The same stacking drawn horizontally	Category range plus a values range per series
Pie	Share of a whole. A Center hole slider turns it into a donut	Category range plus one values range
Waterfall	A running cumulative bridge, colored up and down, for a variance walk	Category range plus one values range
Combo	One visual that mixes column, line, and area series, with an optional second value axis	Category range plus a values range per series, each series carrying its own type and axis setting
Range bars	Floating vertical bands. Each category spans a start value to an end value instead of rising from the axis	One series carrying both a Start (low) range and an End (high) range
Range area	The same bands drawn as a filled corridor across the categories	One series carrying both a start range and an end range
Heatmap	A matrix colored by value, diverging about a neutral point	One rectangle : a matrix block, or a row, column, value table
Sankey	A flow diagram whose ribbon widths are proportional to value	One flow table of source, target, and value, plus an optional node table
Map	Points and value-sized bubbles over map tiles or an image	A marker table, a bubble table, or both. See Maps

Pie and donut

There is no separate donut type. Select **Pie**, then move the **Center hole** slider. At zero the readout reads **None (pie)** and a solid pie is drawn; anything above zero draws a donut. The hole is expressed as a percentage of the pie, up to 90 percent, so the same setting looks right at both the pane's preview size and the imported picture's size.

Combo

Choosing **Combo** reveals two extra controls on every series row:

- A **type** list: **Column**, **Line**, or **Area**.
- A **2nd axis** checkbox, which plots that series against a second value axis.

This is what lets a revenue-in-currency series and a margin-percent series share one chart and both read correctly.

Range bars and range area

A range series is **one** series with two edges rather than two separate series, so the band shares a name, a color, and a legend entry. Bind the **Start (low) range** and the **End (high) range**, and turn on the difference label to annotate each band with its height, meaning the end value minus the start value. The label is calculated from the resolved data, so it can never disagree with the band it sits on.

Reportworq tidies the pairs before drawing, so a messy range does not produce a misleading picture:

- **Ranges of different lengths are padded**, so later bands stay aligned with their own labels.
- **A pair with only one edge is dropped**, rather than drawn down to zero, which would read as an enormous band instead of as missing data.
- **An inverted pair is swapped**, so the start is always the low edge.

Heatmap

A heatmap is the only type that reads a rectangle. Two source shapes are accepted, and both end up as the same matrix:

Shape	What the range holds
Matrix block	A block whose first row is the column headers and whose first column is the row headers, values in the middle.
Row / column / value	Three columns, one row per cell. Columns are matched by header name, falling back to the first three columns.

Create table writes a ready-made block at the active cell, in whichever shape is selected, and binds it.

A heatmap follows these drawing rules:

- **Color diverges about the Neutral point**, which is `0` by default and can be a typed number or a cell. Cells below it are red, cells above it are green.
- **Each side is scaled against its own extreme**, so one large outlier on one side cannot wash out the other side.
- **Rows and columns keep first-seen order**. Your ordering is meaningful, so Jan stays before Feb; nothing is sorted alphabetically.
- **A cell the source never mentions is left blank and outlined, never colored as zero**. On a diverging scale, zero means "on plan", which is a claim the missing data never made.
- **Show each cell's value** prints the number inside the cell. It is on by default and is dropped automatically once the cells get too small to hold it.

Sankey

The whole diagram binds to **one flow table**, where each row is a single ribbon. The nodes are derived from the flows themselves, in first-seen order, and a node's thickness is the larger of its inflow and its outflow.

Binding	What it holds
Flow table (with headers)	One row per ribbon: a source, a target, and a positive value. Columns are matched by header name (<code>source</code> , <code>from</code> , or <code>origin</code> ; <code>target</code> , <code>to</code> , or <code>destination</code> ; <code>value</code> , <code>amount</code> , <code>flow</code> , <code>qty</code> , or <code>count</code>), falling back to the first three columns.
Node table (optional)	One row per node you want to change: the node's name, an optional display label, and an optional color. Rows naming a node that no flow references are ignored.

Create table writes a worked flow table at the active cell and binds it.

Rows that cannot be drawn are dropped before either the preview or the refresh sees them, so both always draw the same graph. A row is dropped when an endpoint is blank, when a row points at itself, or when the value is zero or negative.

The **Appearance** panel holds the presentation options:

Option	Values	Default
Ribbon color	By source, By target, One color	By source
Node alignment	Stretch, Left, Right	Stretch
Show labels , and their placement	On or off; Inside, Before, After	On, Inside
Totals in labels	On or off	Off
Legend	On or off	Off
Node width	2 to 60	12
Node gap	0 to 80	14
Ribbon opacity	0.05 to 1	0.4
Keep the table's node order (no auto-layout)	On or off	Off, so the layout reorders nodes to reduce ribbon crossings

Maps

A map is the one visual driven by *where* your data is, not only by the numbers. It is also the deepest part of the builder, so it is worth understanding in full.

Choose a background

Select **Map background**, then one of two backdrops.

Tile layer is the default. It draws real OpenStreetMap tiles, framed to a **center** and a **zoom** level. Points are positioned by **latitude and longitude**.

Static image draws over a raster image, either one you upload or the built-in **Sample floor plan**. Points are positioned by **X and Y as a percentage, 0 to 100**, of the image. The image does not have to be a map at all: a store floor plan, a warehouse layout, a building schematic, or a process diagram all work, with your data plotted onto them.

Static-image rules to know:

- **PNG, JPEG, GIF, WebP, and BMP are accepted. SVG is refused**, because the renderer that redraws the map on a job run cannot rasterize it.
- **The image is capped at 3 MB**, and it is embedded in the chart's own definition, so a smaller image keeps the workbook lighter.

Add points with marker and bubble layers

Both backgrounds take two data layers under **Data layers**, and each binds to a **table with headers**:

- A **Marker table** drops a labeled pin per row.
- A **Bubble table** adds a value column and sizes each bubble by it.

Columns are matched by header name, so the order of the columns does not matter:

Column	Accepted header names
Position on a tile layer	lat or latitude, and lng, lon, long, or longitude
Position on a static image	x, x%, or left, and y, y%, or top
Label (optional)	label, name, item, or title
Value (bubbles only)	value, amount, size, qty, or quantity

Create table beside either layer writes a ready-made template with the right headers at the active cell and binds it, so you can see the shape before you fill it in.

Three practical notes:

- **Numbers may carry formatting.** Thousands separators, currency symbols, and percent signs are stripped before the value is read.
- **A row with no position and no label is skipped** as blank, so an over-selected range does not produce stray points.
- **Bubbles are not labeled; markers are.** A bubble layer on its own gives you magnitude with no identity. Bind the **same** table range to **both** layers to get sized circles and labeled pins together. A marker label longer than 16 characters is shortened.

Frame a tile map

A tile map can be framed two ways, and both write the same center, zoom, and search values behind the scenes, so switching between them never loses what you set.

Interactive (the default) shows a live map in the pane. Move it, zoom it with its own controls, or use its **Search** box to find a place. Wherever you land is recorded as fixed values.

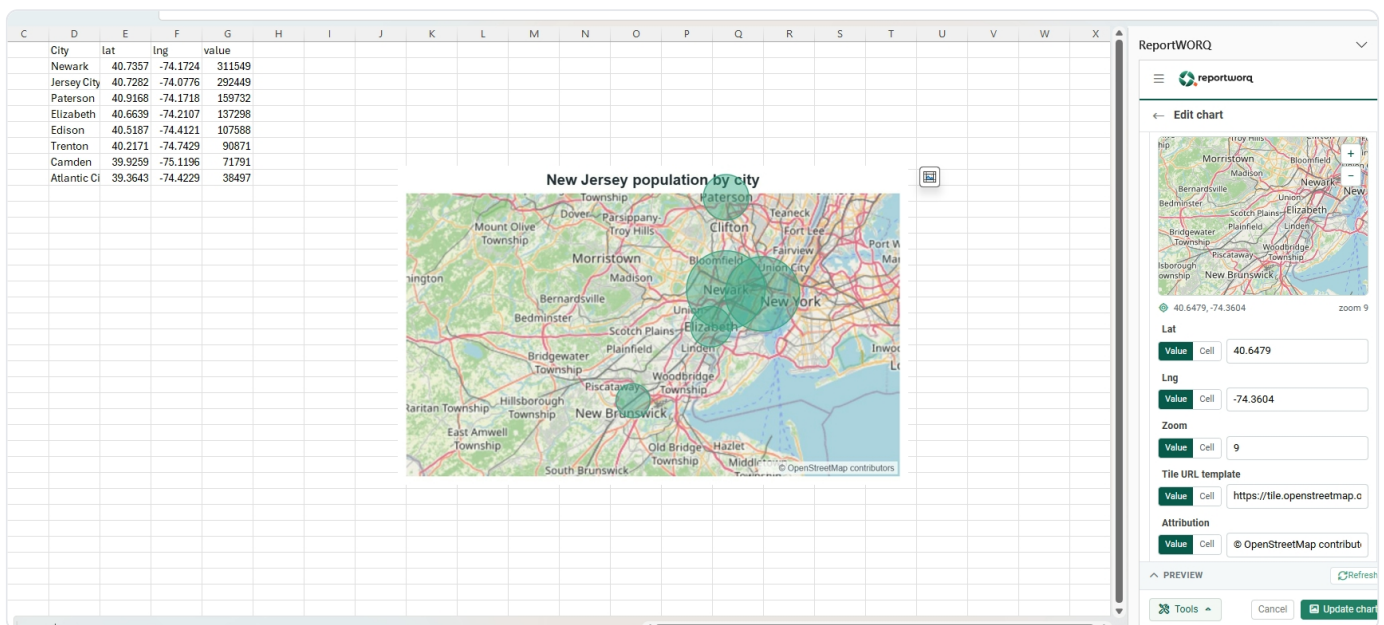
Manual replaces the map with four fields you can type or bind: **Search (auto-center)**, **Lat**, **Lng**, and **Zoom**. Fill them in, then select **Preview**.

Switching to **Interactive** and then moving the map **replaces** any cell-driven or searched framing with fixed values. Reportworq confirms before it does this, so an accidental nudge cannot quietly undo a data-driven map.

Drive the map from your data

This is the technique that makes maps worth building in Reportworq, and it is what turns one authored map into a whole burst set.

Every framing field carries a **Value | Cell** toggle: **Search (auto-center)**, **Lat**, **Lng**, **Zoom**, and even the **Tile URL template** and **Attribution**. Set a field to **Cell** and it reads a worksheet cell instead of holding a fixed value. Because a cell can hold a formula or a looked-up value, the map **frames itself from your data**, and reframes on every run.



Which field wins. If both **Lat** and **Lng** resolve to numbers, they position the map and any search term is ignored. Otherwise **Search** centers it. The two are not mutually exclusive, so you can set both and let that rule decide.

That gives you two complementary ways to say where the map should look, and a third field that decides how close it gets.

1. You have coordinates. Bind **Lat** and **Lng** to cells holding latitude and longitude. This is the most precise option and the natural fit when your data already carries coordinates.

2. You only have a place name, so search for it. Bind **Search (auto-center)** to a cell instead. If your data says *revenue in New Jersey* or *headcount in Newark*, put **New Jersey** or **Newark** in a cell and bind **Search** to it. Reportworq looks the place up and

centers the map on it, with no coordinates required. Because the search term is only a cell, it is fully data-driven: change the cell, or let a burst supply it, and the map moves.

3. Zoom from a lookup table. Search finds a place, but it does not decide how close to get. Zoom is resolved independently, and an unset zoom falls back to a wide default view whatever you searched for. So bind **Zoom** to a cell as well.

The real payoff is a small lookup table of your own that pairs each location with the zoom that frames it best. Keep it on the worksheet, look up the current location's row with an ordinary Excel lookup, and bind **Zoom** to the result:

Location	Zoom
North America	3
Northeast region	6
New Jersey	8
Newark metro	11
Ironbound district	14

Now a run for a dense city zooms in tight while a run for a sprawling region pulls back, each framed correctly, without hand-tuning every map.

Put the three together and one map definition serves a whole burst set. The burst supplies the location into a cell, **Search** recenters the map on it, the zoom lookup picks the right altitude, and the marker or bubble layer redraws from that location's rows. Every recipient gets a map framed for their own place, from a single authored chart. See [Burst sets](#).

Rules for map cell references

Map framing fields are stricter than the other bindings, because each one resolves to a single value rather than a range:

- A framing field must reference a **single cell**, not a range or a named range.
- That cell must be on the **current sheet**. A reference to another sheet is refused with a message naming the sheet you are on.

Category, series, title, and the marker and bubble tables are not restricted this way. They may point at another worksheet.

Tile source (advanced)

Tile source (advanced) is collapsed by default and rarely needs changing. It holds two fields, both of which also take a **Value | Cell** toggle:

- **Tile URL template**, which defaults to the standard OpenStreetMap raster tiles.
- **Attribution**, the credit line drawn in the corner of the map, which defaults to the OpenStreetMap contributors credit.

Change these only if your organization serves its own map tiles. The attribution credits the tile service, so change that text only when you have changed the tile source.

Map notes and limits

- **A tile map is a static image inside Excel.** You frame it live in the task pane; the sheet holds the stitched picture, not a map you can pan.
- **Zoom is a whole number.** The field suggests 2 to 18. A fractional value is rounded, and a value outside the supported range is brought back into it.
- **A point outside the framed area is not drawn**, and nothing warns you. Choose the zoom against the actual spread of your points, not only the center.
- **Real tiles and place-name search both need outbound HTTPS from the server** to the tile and lookup services, at preview time and again at refresh time. If they are unreachable, the map falls back to a plain grid canvas with your marker and bubble layers intact and correctly positioned.
- **A failed place lookup keeps the previous center rather than reporting an error.** A map centered on the wrong place looks exactly like one that worked, so check the picture after a run that introduced a new location.

How a chart stays current

An imported chart or map is a **picture that carries its own definition**, so a refresh reads the definition back, re-reads the bound ranges, and redraws.

In Excel, refresh is something you ask for. The picture is a visualization, not a live Excel chart object: it does not respond to pointing or selection, and it does not redraw as you type. Use **Refresh chart** on a single chart, or the workbook-level **Refresh**, to bring it up to date.

On a job run it is automatic. Reportworq redraws **every** Reportworq chart on the workbook from the current cell values, with no one opening Excel. A series or a map field bound to a formula reads the **calculated** result, so charts see the same numbers the rest of the report does.

Three further behaviors follow from the design:

- **Resize the picture, then refresh, to redraw at the new dimensions.**
Stretching the picture on its own scales the existing image; refreshing redraws the visual at the size the picture now occupies, which corrects the proportions and sharpens it.
- **Job output is drawn at three times the on-sheet size**, so a chart stays sharp in a PDF, in a PowerPoint deck, and on screen when someone zooms in.
- **A replicated worksheet's charts follow their own copy.** When a worksheet is replicated for a burst, one copy per parameter value, each copy's chart re-points at its own sheet, so every page shows its own data rather than repeating the first page's. A binding you deliberately pointed at a *different* sheet is left exactly as authored.

Authoring aids

Tools at the foot of the builder holds three helpers.

Tool	What it does
Create with AI	Select a block of data and Reportworq works out which column is the category and which are the series, then fills in the bindings. Needs a configured AI model; without one a fixed rule is used instead.
Sample Data	Writes a ready-made example dataset onto the sheet and configures the form to plot it, so you can see the shape a type expects. The samples cover quarterly revenue, sales by region, revenue by segment, channel mix, stock price, market share, cash flow, revenue and margin, sales pipeline, New Jersey population on a tile map, and a grocery floor plan image map.
Use named ranges	Converts every cell and range binding on the current screen into workbook named ranges, so the chart keeps pointing at the right data when rows or columns are inserted or deleted. It confirms first, lists what it will convert, and tells you when there is nothing to convert.

Create table appears beside the bindings that expect a table of a specific shape, on maps, Sankey, and heatmap. It writes a correctly headed template at the active cell and binds it.

Notes and limits

- **Charts and maps are imported as pictures, not live Excel chart objects.**
This surface is a prototype: imported visuals are not yet versioned or permissioned as report objects the way a report is.
- **A chart depends on its bound ranges still holding what it expects.** If a report import changes the shape of the data underneath a chart, re-check the

binding. **Use named ranges** makes this less likely.

- **Category axis labels are shortened to about eight characters** when they are drawn, and legend entries to about 14. Keep the labels on the chart short; the data sheet can keep the full names.
- **A chart can live on a different worksheet from its data.** Bind through the range picker, which returns a sheet-qualified address, or through a workbook named range.
- **A definition that cannot be read is skipped silently**, leaving the previous image in place. If a picture stops updating while others refresh, rebuild it from the builder.

Going deeper. For how these visuals compare with native Excel charts, and when to reach for each, see [Charts and maps](#). The same add-in loads in PowerPoint, where a slide shape can be linked to a chart in the Excel report, see [PowerPoint reporting](#). For how a refreshed report is distributed, see [The report lifecycle](#).

Workday Adaptive functions

reference

report author

The Workday Adaptive family reads data from, and writes data back to, **Workday Adaptive Planning**. Each function has a canonical `RW.Adaptive.*` name and a legacy `AI*` alias.

Function	Alias	What it does
<code>RW.Adaptive.Extract</code>	<code>AIEXTRACT</code>	Builds and imports a custom Adaptive report from tier and option formulas
<code>RW.Adaptive.Tier</code>	<code>AITIER</code>	Defines one row, column or filter axis for an Extract
<code>RW.Adaptive.Option</code>	<code>AIOPTION</code>	Sets a named report option that Extract reads
<code>RW.Adaptive.Filter</code>	<code>AIFILTER</code>	Stores a reusable dimension filter
<code>RW.Adaptive.Input</code>	<code>AIINPUT</code>	Marks a cell for upload to Adaptive on submit
<code>RW.Adaptive.ModeledInput</code>	<code>AIMODELEDINPUT</code>	Updates one field on a modeled Adaptive sheet
<code>RW.Adaptive.ModeledExport</code>	<code>AIMODELEDEXPORT</code>	Exports modeled Adaptive data into a range or table

The `AI*` alias is not artificial intelligence

The aliases in this family begin with `AI`: `AIEXTRACT`, `AITIER`, `AIOPTION`, `AIFILTER`, `AIINPUT`, `AIMODELEDINPUT`, `AIMODELEDEXPORT`. That `AI` is short for **Adaptive Insights**, the former name of Workday Adaptive Planning. These functions have **nothing to do with generative AI or large language models**. They are a data connector.

The only generative worksheet function is `RW.Tools.AIInsight`. See [The AI insight function](#). Do not confuse the two.

Before you use these functions

- A **Workday Adaptive Planning connection** with valid metadata must be configured for the workbook. An admin sets this up; see the [Connector catalog](#).
- The read and export functions run during **report generation**. The input and modeled-input functions run during **input submission**, not the output pass.
- The functions are inert in desktop Excel.

- Version availability across this family is approximate: most date from at least around v5.0.0.42, with `RW.Adaptive.Filter` from around v5.0.0.79.

RW.Adaptive.Extract

Generates an Adaptive report into a worksheet range by combining the tier formulas that define its axes with the option formulas that control its output.

Signature

```
=RW.Adaptive.Extract(targetRange, tiersRange, optionsRange)
```

#	Argument	Type	Meaning
1	<code>targetRange</code>	Range	Top-left output range for the report.
2	<code>tiersRange</code>	Range or multi-range	The cells holding the <code>RW.Adaptive.Tier</code> formulas for rows, columns and filters.
3	<code>optionsRange</code>	Range or multi-range	The cells holding the <code>RW.Adaptive.Option</code> formulas.

All three are required.

Behavior

- Extract gathers every `RW.Adaptive.Tier` formula from `tiersRange` and splits them into rows, columns and filters. **At least one row tier and one column tier are required.**
- `RW.Adaptive.Option("AutoFormat", ...)` controls whether Reportworq bolds the row and column labels and auto-fits the label columns.
- The Adaptive result set is imported into `targetRange`, and the formula cell is cleared on success.
- During the Adaptive pass, **modeled exports run first**, and Extract runs afterwards, once the workbook's Adaptive specs have been expanded.

Keep the tier and option formulas in small, clearly labeled setup ranges and pass those ranges in. Control formatting and suppression through `RW.Adaptive.Option` rather than hard-coding it into the template.

Example

```
=RW.Adaptive.Extract(A1, Setup!B2:B6, Setup!D2:D4)
```

RW.Adaptive.Tier

Defines one Adaptive tier for the row axis, column axis, or filter axis of an Extract.

Signature

```
=RW.Adaptive.Tier(axis, dimensionName, selections, [expansionType], [order])
```

#	Argument	Type	Default	Meaning
1	axis	String	<i>required</i>	row or rows; col, cols, column or columns; filter or filters.
2	dimensionName	String	<i>required</i>	The Adaptive dimension, resolved by name in Adaptive metadata.
3	selections	Range or string	<i>required</i>	The members for this tier. A single text value containing ; is split into multiple members.
4	expansionType	String	"none"	none, children, descendants, or leaf.
5	order	Integer	0	Sequences multiple tiers that share an axis.

Behavior

- With `expansionType="none"` at least one element must be supplied.
- With `children`, `descendants` or `leaf`, the tier can expand from the members you supply **or from the dimension root** when no members are supplied.
- Extract sorts row and column tiers by `order` before building the request. Put outer tiers at lower `order` values and inner tiers at higher ones.
- Use `leaf` when you want fully expanded leaf-level members rather than parents.
- Use `filters` for dimensions that should constrain the report without appearing as row or column labels.

Examples

```
=RW.Adaptive.Tier("rows", "Period", A1:A5)
=RW.Adaptive.Tier("columns", "Region", B1:B2, "children", 1)
=RW.Adaptive.Tier("filters", "Version", "Actual;Budget")
```

When an argument is wrong

Situation	Result
Invalid axis name	The tier is not assigned to rows, columns or filters correctly.
<code>expansionType="none"</code> with no selections	Formula error.
A member name does not exist in the dimension	Report generation fails when the tier is expanded.

RW.Adaptive.Option

Defines an option that Extract reads when building the report.

Signature

```
=RW.Adaptive.Option(optionName, optionValue)
```

#	Argument	Type	Meaning
1	<code>optionName</code>	String	An option name such as <code>"AutoFormat"</code> , <code>"SuppressZeros"</code> or <code>"SuppressRollups"</code> .
2	<code>optionValue</code>	String, Boolean or integer	The value stored with the option.

Reportworq stores the value as text and also tries to parse it as a Boolean and as an integer. The formula cell displays `optionName=optionValue`.

Use `SuppressRollups` and `SuppressZeros` to shape the Adaptive request itself, rather than post-processing the result in Excel.

Examples

```
=RW.Adaptive.Option("AutoFormat", TRUE)
=RW.Adaptive.Option("SuppressZeros", 2)
=RW.Adaptive.Option("SuppressRollups", FALSE)
```

When an argument is wrong. A misspelled option name is **silently unused** by Extract, so an option that appears to have no effect is most often a typo. A numeric option whose value is text that cannot be parsed as an integer also fails quietly.

RW.Adaptive.Filter

Stores a reusable Adaptive dimension filter: a dimension name and one or more member names.

Signature

```
=RW.Adaptive.Filter(dimensionName, memberNames)
```

#	Argument	Type	Meaning
1	dimensionName	String	The dimension to filter, such as "Version" or "Region".
2	memberNames	Range, array or literal list	The members to keep.

Literal lists and cell-based lists are both supported. The formula cell displays a `dimension.members` summary.

It defines filter values only. It does not render a report grid on its own; for the full row, column and filter layout use `RW.Adaptive.Tier` and `RW.Adaptive.Extract`.

Examples

```
=RW.Adaptive.Filter("Region", A1:A5)
=RW.Adaptive.Filter("Version", {"Actual","Plan"})
```

RW.Adaptive.Input

Marks a cell as an Adaptive input and prepares it for upload during contribution processing.

Signature

```
=RW.Adaptive.Input(inputCell, [adaptiveSheetName], [tuplePart1], [tuplePart2], ...)
```

#	Argument	Type	Default	Meaning
1	<code>inputCell</code>	Cell reference	<i>required</i>	The cell whose value is uploaded.
2	<code>adaptiveSheetName</code>	String	<i>none</i>	When supplied, the upload is treated as a cube account upload for that Adaptive sheet. When omitted, it is a standard account upload.
3 and up	<code>tuplePart</code>	String	<i>optional</i>	Explicit tuple coordinates in `Dimension

Behavior

- Supply explicit tuple parts when you want the formula itself to define where the value is written. Leave them out when the generated workbook already carries the tuple map Reportworq created for that input cell during report generation.
- Processed during input submission, not the output pass. The formula cell is cleared on success.

Examples

```
=RW.Adaptive.Input(B5)
=RW.Adaptive.Input(B5, "CommissionSheet")
=RW.Adaptive.Input(B5, "CommissionSheet", "Version|Plan", "Period|2026-01", "Region|East")
```

When an argument is wrong

Situation	Result
<code>inputCell</code> is not a real cell reference	Formula error during parsing.
A tuple part is not in `Dimension	Member` form
The referenced tuple cannot be found in workbook metadata	Submission fails for that input.

RW.Adaptive.ModeledInput

Updates a modeled Adaptive sheet by matching one key value and writing one modeled field value.

Signature

```
=RW.Adaptive.ModeledInput(version, adaptiveSheetName, importKeyName, importKeyValue,
inputValueName, inputValue)
```

#	Argument	Type	Meaning
1	version	String	The Adaptive version to update.
2	adaptiveSheetName	String	The modeled sheet to update.
3	importKeyName	String	The key column used to find the row.
4	importKeyValue	Value	The key value to match.
5	inputValueName	String	The modeled field or column to update.
6	inputValue	Value	The new value to write.

All six arguments are required. The formula builds a small two-column modeled-data update request and sends it during input processing. The formula cell is cleared on success.

Use `INDIRECT(...)` when you copy the formula down repeated worksheet rows, so the references track the row rather than drifting.

Use `RW.Adaptive.Input` instead for standard or cube-account uploads that rely on tuple metadata.

Examples

```
=RW.Adaptive.ModeledInput($C$2, "CommissionSheet", "Employee_ID", A2, "Target_Commission",
B2)
=RW.Adaptive.ModeledInput("2026_Plan", "Targets", "Region", INDIRECT("A" & ROW()),
"Headcount", INDIRECT("B" & ROW()))
```

When an argument is wrong

Situation	Result
Any required argument is blank	Formula error before submission.
<code>adaptiveSheetName</code> does not match a modeled sheet	Submission fails.
Used for normal tuple-based input cells	Use <code>RW.Adaptive.Input</code> instead.

RW.Adaptive.ModeledExport

Exports modeled Adaptive data into a worksheet range or named table.

Signature

```
=RW.Adaptive.ModeledExport(targetRange, version, [adaptiveSheetName], [startPeriod],  
[endPeriod], [levels], [includeDescendent], [isGlobal])
```

#	Argument	Type	Default	Meaning
1	targetRange	Range or table	<i>required</i>	Output location.
2	version	String	<i>required</i>	The Adaptive version to export.
3	adaptiveSheetName	String	<i>none</i>	Optional modeled sheet name.
4	startPeriod	String	<i>none</i>	Optional start-period filter.
5	endPeriod	String	<i>none</i>	Optional end-period filter.
6	levels	String	<i>none</i>	A pipe-delimited level list such as <code>`"Level1</code>
7	includeDescendent	Boolean	FALSE	Include descendant members.
8	isGlobal	Boolean	FALSE	Request a global export.

Behavior

- The export can be written into a normal range or into a named Excel table. A table is useful when downstream formulas, filters or charts should grow with the result set.
- Modeled exports run **before** Extract during the Adaptive formula pass.
- If the export returns rows, the data is imported and the formula cell is cleared.

Examples

```
=RW.Adaptive.ModeledExport(A1, "ActualData")  
=RW.Adaptive.ModeledExport(SalesTable, "PlanModel", "CommissionSheet", "2026-01", "2026-12")  
=RW.Adaptive.ModeledExport(B5, "ForecastData", , , , "Level1|Level2", TRUE, FALSE)
```

When an argument is wrong

Situation	Result
Invalid target range or table name	Formula error.
<code>levels</code> separated by commas	Not recognized. The implementation splits on `
Omitting placeholder commas for skipped optional arguments	The later arguments are read into the wrong positions. See the third example.

Use cases

- **Adaptive custom report.** `RW.Adaptive.Tier` formulas define Period rows and Region columns, `RW.Adaptive.Option("SuppressZeros", TRUE)` trims the empties, and `RW.Adaptive.Extract` renders the grid for a monthly management pack.
- **Modeled forecast package.** `RW.Adaptive.ModeledExport` pulls a modeled sheet into a table for a forecast workbook, and after review `RW.Adaptive.ModeledInput` writes the revised targets back.
- **Commission upload.** `RW.Adaptive.Input` marks the commission cells for cube-account upload during a contribution round.

Notes and limits

- The functions are inert in desktop Excel. They only run inside a Reportworq job or form submission.
- Extract needs at least one row tier and one column tier.
- Remember the naming trap: the `AI*` aliases are Adaptive functions, not AI.

Related pages

- [Functions overview: which one do I use](#)
- [General report and layout functions](#) for the layout functions that run alongside an Extract.
- [The AI insight function](#) for the genuinely generative one.
- [Connector catalog](#) for the Adaptive connection setup.

When a job fails

[how-to](#) [report author](#) [administrator](#)

When a run fails or completes with warnings, Reportworq keeps a per-run diagnostics archive so you can see exactly what happened. Read the log to see what went wrong, work through the common causes, and, when needed, escalate to support with the right logs. This page shows how to find the cause and escalate.

When to use it. Any time a run shows **Failure** or **Warnings**, or a job appears to succeed but produces no output. Start here before re-running blindly.

Open the diagnostics for a failed run

1. Find the failed run in [Activity](#) (filter to **Failures**) or in the report's [Run History](#).
2. From Activity, select **Details** on the run. From Run History, open the failed run; its error card offers **Run again** and **Send logs to support**.
3. This opens the diagnostics screen. It offers **Diagnostics**, which holds the log, attachments, and trace; **Performance**, which is present on every run (see [Review a run's performance](#)); and **AI diagnostics**, which appears only for a run that failed or warned. It opens on **Diagnostics**.

Workspace > Timberline > Financial Reports > Financial Statements - Job Failure

Financial Statements - Job Failure

Run history

Run now

Run history. Every run of this report - newest first. Each run shows the outputs it produced, their status, the parameters used, and the formats delivered. Open a run to inspect outputs, downloads, and how each was distributed.

Runs are kept for 30 days (system default); older runs are removed automatically. Pin a run to keep it forever.

All 1Failed 1

	STATUS	STARTED	ELAPSED VS BASELINE	TREND	PEAK MEM	OUTPUTS	FORMATS	DELIVERY
	Failed	Jul 31, 2026 - 20:20	13.6s		300 MB	1 variant - 2 files	PDFXLSX	Look into error

1 run is held out of the baseline (failed, cancelled, or no telemetry), so a broken run can't drag the comparison.

Read the log

1. On the **Log** tab, read the entries top to bottom. Each has a line number, timestamp, elapsed time, and message. An error entry expands to show the exception message and stack trace.
2. The first 250 entries load; select **Show Full Log** to see the rest.
3. The **Attachments** tab lists any files the run produced (viewable and downloadable). The **Trace** tab holds HTTP traces, which are present only if **Trace logging** was on for the run. To capture traces, re-run the job with Verbose and Trace logging enabled (see [Run a job](#)).

The screenshot shows the reportworq interface. On the left is a sidebar with 'Workspace' and various tools like 'My Input Forms', 'Scheduled Content', 'Global Parameters', 'Address Book', and 'Data Collection'. The main area displays a job titled 'Financial Statements - Job Failure' with a status of 'Failed'. The error message is 'Distribute: email Failed'. Below the error message, there is a 'TRACE' section showing 'Distribute: email Failed' and 'Error sending email to'. A dropdown menu is open, showing options: 'View log summary', 'Export History', 'Send to support', and 'Advanced Diagnostics'. At the bottom, there is an 'OUTPUTS' section showing '2026-USA - South Financial Statements' with 2 files (PDF, XLSX). The status is 'Failure - 94 datapoints'.

Common causes to check

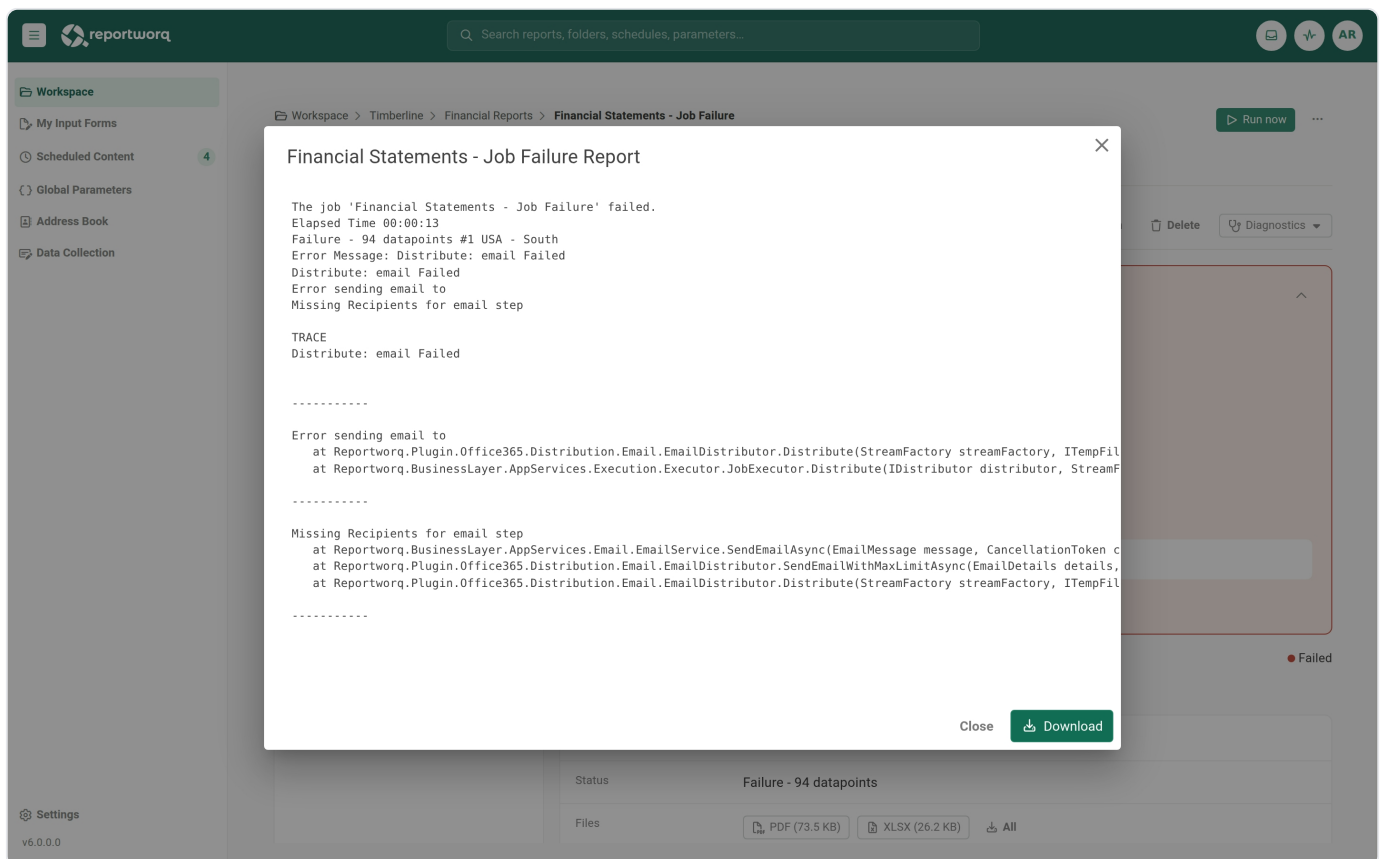
- **A data source is unavailable or timed out.** The log or Trace tab shows the failing call. Confirm the source system is up and the connection still works.
- **A parameter resolved to nothing.** A parameter value that returns no data can leave a run with no output. Check the parameter and its source query.
- **No distributor for the output.** A run can generate output but deliver nothing if the job has no configured distribution. Check the job's distribution.
- **A missing or moved source file.** If a source file was renamed or moved, re-link it in the Job Editor.

- **The run succeeded but produced no output.** Check sheet suppression and any exception-check send-gate on the report, which can intentionally hold or empty the output.

Escalate to support

Use **Send to support** on the run's diagnostics, or **Send logs to support** on a failed run's error card in Run History, to upload the run's archive directly. To attach the logs to a ticket yourself instead, use **Export logs**, which downloads the run's whole execution database as a `.zip`.

If the log does not show enough to work with, re-run the job with **Verbose logging** on, and **Trace logging** as well when a data source is involved, then escalate that run. See [Run a job](#).



Notes and limits

The Trace tab is empty unless Trace logging was enabled at run time, and a log requires job-history logging to have been on for that run.

Going deeper. For symptom-to-fix guidance across the product, see the [Troubleshooting index](#). A run that finished but took far too long is a different investigation, see [Review a run's performance](#).



Contribution functions

reference

A contribution form's behavior is authored in Excel. A workbook designer places **RW.Contribution.*** formulas in the source worksheet; when the campaign generates forms, Reportworq reads the formula metadata and renders the matching editable cells, validation markers, and controls in the contributor's browser form. This page is the reference for those five functions and the placement rules that govern them.

There are five functions, each with a short alias:

Function	Alias	Renders as
RW.Contribution.Input	RW.INPUT	Editable input cells
RW.Contribution.Check	RW.CHECK	Validation marker (red or blue)
RW.Contribution.Checkbox	RW.CHECKBOX	Interactive checkbox
RW.Contribution.List	RW.LIST	Dropdown of allowed values
RW.Contribution.Filter	RW.FILTER	Filter control

Write-back formulas belong in hidden rows

Before the RW.Contribution functions, note the separate family that actually **posts** values to the target system when a workbook is processed through data collection. The formula depends on the target:

Target system	Write-back formulas
IBM Planning Analytics / TM1	DBS , DBSW , DBSS
SQL	RWSQLUPDATE , RWSQLUPsert
Workday Adaptive	AIINPUT , AIMODELEDINPUT
Anaplan	ANAPLANINPUT

Keep these write-back formulas in hidden rows or columns. They render as **#NAME** in the contributor's form, so their rows and columns are hidden by design. Unhiding them exposes broken-looking cells to the contributor. This is a deliberate layout rule, not a defect.

A contribution campaign does not need these formulas. A campaign's collected values reach the target through its own **export**, which maps the data model declared in

`RW.Contribution.Input` to the destination. These formulas are what a returned workbook uses when data collection processes it. See [Data collection processing](#).

DBS, DBSW, and DBSS (Planning Analytics)

The three Planning Analytics write-back functions share one signature and differ only in what they may write.

Signature

```
DBS(value, "server:cube", element1, element2, ...)  
DBSW(value, "server:cube", element1, element2, ...)  
DBSS(value, "server:cube", element1, element2, ...)
```

Argument	Meaning
value	The value to post. A numeric value for <code>DBS</code> and <code>DBSW</code> ; a string for <code>DBSS</code> .
"server:cube"	The Planning Analytics server and cube, as one colon-separated string.
element1, element2, ...	One element per cube dimension, in the cube's dimension order. A single range covering the elements is also accepted in place of the individual arguments.

The three names differ in the cell type each may update.

Function	Writes	Notes
<code>DBS</code>	Numeric cube cells	Rejects a string cell: "DBS does not support updating string cube values."
<code>DBSW</code>	Numeric cube cells	Subject to the same consolidated-cell rule as <code>DBS</code> .
<code>DBSS</code>	String cube cells	Rejects a numeric cell: "DBSS does not support updating numeric cube values." Use it for comments, justifications, and any other text a plan stores as a string measure.

Behavior

- **The element count must match the cube exactly.** Too few or too many elements for the cube's dimension list is an error, so a formula written against an older version of a cube fails rather than posting to the wrong intersection.
- **An empty value is not skipped.** A numeric write with no value posts `0` ; a string write with no value posts an empty string.
- **Consolidated cells are rejected by default.** A numeric write to a consolidated intersection fails with "Consolidated input not supported," matching the Planning

Analytics Excel add-in. Turning on **Proportional Writeback** sends the value with the proportional-spread marker instead, so Planning Analytics allocates it across the leaf cells. `DBSS` is unaffected, because string cells do not consolidate.

- **Match the function to the measure.** Use `DBS` or `DBSW` for numeric measures and `DBSS` for string measures. A cube that stores a comment or a justification alongside its numbers needs `DBSS` for that column.

RW.Contribution.Input

Declares the editable range and its dimensional context. Input cells are created only where every referenced field resolves to a non-blank value, and the formula cell is cleared after evaluation.

Signature

```
RW.Contribution.Input(inputRange, formName, [allowEditing=TRUE], fieldName, fieldRefOrValue, ...)
```

Argument	Meaning
<code>inputRange</code>	The editable cell range. Accepts a multi-block form for non-contiguous regions, for example <code>(D4:D6,D9:D11,F4:G6,F9:G11)</code> .
<code>formName</code>	The logical form (data model) name. This is the Data Model an export selects, so it is not just a label.
<code>allowEditing</code>	Optional, default <code>TRUE</code> .
<code>fieldName,</code> <code>fieldRefOrValue</code>	Repeating pairs that supply the dimensional context. Each <code>fieldName</code> becomes a <code>Fields.*</code> token an export can map to a cube dimension or a table column.

Reportworq classifies each field reference by its shape: a **single-row range** is a row axis, a **single-column range** is a column axis, a single cell or rectangular range is a **filter**, and a literal is a **static value**.

Limits

- The `inputRange` **cannot include control cells** (cells holding filter, checkbox, or picklist formulas). Overlapping the two corrupts both.
- The formula must not sit on a header row or header column (see [Placement rules](#)).
- **One data model name must carry one field list.** If two forms in a campaign declare the same `formName` with different `fieldName` values, the export cannot resolve the model and reports that it is not consistent across forms.

RW.Contribution.Check

Adds a validation rule for a target cell. The formula cell evaluates to the Boolean result.

Signature

```
RW.Contribution.Check(targetCell, validationPassed, validationMessage,
[validationRequired=FALSE])
```

Argument	Meaning
targetCell	The cell the rule validates.
validationPassed	A Boolean expression; TRUE means the value is valid.
validationMessage	The message shown when the rule fails.
validationRequired	Optional, default FALSE . A required rule (TRUE) is a hard stop.

Behavior

- A **required** rule renders as a **red/blocking** marker and prevents Submit until it passes. A **non-required** rule renders as a **blue/warning** marker and is advisory.
- Validation **runs on every value change**.
- **Multiple checks can target one cell**.
- Failing messages surface **both as cell comments and on the Validations tab** of the Details pane.

RW.Contribution.Checkbox

Creates a checkbox bound to a target cell that stores the checked value.

Signature

```
RW.Contribution.Checkbox(targetCell, label)
```

The formula cell shows a [] label placeholder in Excel; the rendered form turns it into an interactive checkbox. The targetCell must be a real cell reference, not a literal.

RW.Contribution.List

Creates a dropdown bound to a target cell.

Signature

```
RW.Contribution.List(targetCell, listValuesRange, includesHeader)
```

Reportworq reads the range, removes duplicates, and stores the option list. With `includesHeader=TRUE`, the first value is treated as a header, not a choice. The formula cell evaluates to a `Picklist: ...` summary. The `targetCell` must be a real cell reference.

RW.Contribution.Filter

Defines a filter control and its value list. The formula cell evaluates to the label text.

Signature

```
RW.Contribution.Filter([label], [filterValuesRange])
```

Arguments given	Behavior
0	Reads values from the cells directly below the formula, stopping at the first blank.
1 (<code>label</code>)	Sets an explicit label and still auto-detects values below.
2 (<code>label</code> , <code>filterValuesRange</code>)	Takes an explicit value range, which must be a single column .

The rendered control has a search box; applying criteria relabels **Select All** to **Select All Selected**, multiple criteria can be selected, and the filter applies to **input-form rows only**.

Signatures at a glance

Function (alias)	Key arguments	Notes
RW.Contribution.Input (RW.INPUT)	inputRange, formName, [allowEditing=TRUE], fieldName, fieldRefOrValue, ...	Axis inferred from each reference shape; blank-resolving cells skipped; formula cell cleared; multi-block range allowed; no control cells inside the range.
RW.Contribution.Check (RW.CHECK)	targetCell, validationPassed, validationMessage, [validationRequired=FALSE]	Required = red/blocking; non-required = blue/advisory; runs on every change; multiple per cell.
RW.Contribution.Checkbox (RW.CHECKBOX)	targetCell, label	Value stored in target cell; shows [] label placeholder.
RW.Contribution.List (RW.LIST)	targetCell, listValuesRange, includesHeader	Duplicates collapsed; first value is a header when includesHeader=TRUE .
RW.Contribution.Filter (RW.FILTER)	[label], [filterValuesRange]	0/1 arg auto-detects values below (stops at first blank); explicit range must be single-column.

Adjacent input function: ANAPLANINPUT

Beyond the RW.Contribution family, `ANAPLANINPUT(inputCell, [tuplePart...])` also marks a cell as a contribution write-back input, flagging its value to post to the matching Anaplan module cell during processing. It is Anaplan-connector-specific and lives in the broader Excel-functions catalog, not the RW.Contribution family.

Workbook design rules

The function signatures are only half the job. These placement and layout rules are where forms actually break:

- **Never place an input, validation, or filter formula on a header row or header column.** Do not put a formula in a row that contains column headers, or a column that contains row headers. The axis inference in RW.Contribution.Input depends on the formula sitting *inside* the data block; on a header line it misroutes or breaks the classification.

- **An input range cannot include control cells** (filter, checkbox, or picklist formula cells). Overlapping them corrupts both.
- **Print Area governs the form's extent.** With a Print Area set, only cells inside it appear, and a group that extends past the boundary is truncated, except a chart that spans the boundary is included in full. With no Print Area set, the form is a rectangular block of all non-empty cells, and grouped rows or columns are included whole even if the group contains empty rows.
- **Excel data filters exclude hidden rows** from the form.
- **Merged cells that span a freeze-pane border anchor to the frozen pane.**
- **Sparklines are supported** and update when their linked input cells change.
- **Validation surfaces twice:** every failing check appears both as a cell comment and on the Validations tab of the Details pane.

Availability

Every function on this page is available in the current release of Reportworq. The RW.Contribution family was completed over a series of Reportworq 5 updates, `RW.Contribution.Filter` last, so a workbook authored against an older Reportworq 5 build may not support the full set. If you are targeting an older installation, confirm the version before relying on the newer controls.

Related

- [Author a contribution campaign](#) generates forms from the workbook these formulas live in.
- [Fill in and submit a form](#) shows how the rendered controls and markers behave for a contributor.
- [Contribution \(write-back\) overview](#) sets the mental model.
- [Data collection processing](#) covers the separate path that runs the hidden write-back formulas.

Advanced parameters: variables and cascading

[explanation](#)[report author](#)

Parameters are usually independent: each supplies its own value. Their real power, though, comes from letting **one parameter reference another parameter's value as a variable**, so a query runs differently, or repeats, based on what a second parameter holds. This is how a report fans out one slice per item, one report per element, or a whole set of nested results, from a single template.

This page explains the mental model and the tokens behind it. For the task steps, see [Parameterize a report](#); for the type catalog and replication modes, see the [Parameters reference](#).

The core idea: a parameter inside a parameter

Some parameter types carry a **query**, SQL Query, Planning Analytics MDX Query, and Cube Value. Inside that query text you can embed a **token** that stands for another parameter's value. At runtime Reportworq substitutes the token before running the query. Because the query's results become the referencing parameter's own values, a single parameter can be driven by another, and can repeat.

Only **query-bearing** types scan for these tokens. Plain Text Value and Text List parameters do not.

The three tokens

`%param:Name%`, another parameter's value

Write `%param:Name%` (or the equivalent `%param_Name%`) anywhere in a SQL, MDX, or Cube query, and Reportworq replaces it with the value of the parameter named `Name` before the query runs. This works in **all** query-bearing types, SQL Query, MDX Query, and Cube Value, not only SQL.

Example: a `Departments` SQL parameter whose query is `SELECT dept_code FROM departments WHERE region = '%param:Region%'` returns the departments for whichever region the `Region` parameter currently holds.

`%global:Name%`, a workspace global's value

`%global:Name%` (or `%global_Name%`) substitutes the value of a workspace or global parameter the same way. Use it for values that are set once at the workspace level, such as a current fiscal year or a default currency, and shared across many reports.

Repeating and fan-out

Embedding a variable is what makes a query **repeat**. There are two ways this shows up.

Cascading, one report per item

Put `%param:Region%` in a SQL or MDX parameter's query, then set **that** parameter's replication to **One report per item** (see [Parameters reference](#)). The query returns one element per item for the referenced region, and Reportworq produces **one output file per element**. This is the cascading, or dependent, parameter workflow: pick a region, and get a separate report for each department in that region.

Multi-value concatenation (Cartesian)

When the **referenced** parameter itself holds **multiple** values, the referencing query is **run once per referenced value, and the result lists are concatenated**.

Reportworq builds the Cartesian product of the referenced values, runs one query per combination, then unions the rows.

Example: if `Region` holds `North`, `South`, and `West`, a `Departments` query that references `%param:Region%` runs three times, once per region, and the returned departments are combined into one list. Add a second referenced parameter and the combinations multiply accordingly.

When to use each

- `%global:Name%`, for a value set once for the whole workspace (fiscal year, reporting currency, company code) that many reports should share without re-entering it.
- `%param:Name%` in a query, when a parameter's allowed values depend on another parameter. Choosing a region should narrow the department list to that region.
- **Cascading, one report per item**, when you want a separate, self-contained output file per element that a dependent query returns, for example one department report per department in the chosen region.

Notes and limits

- Tokens are **literal substitution**. Reportworq replaces the exact token string before the query runs, so the surrounding query must remain valid after substitution (mind the quoting).
- Both token forms are accepted: `%param:Name%` and `%param_Name%` are equivalent, as are `%global:Name%` and `%global_Name%`.
- Cross-parameter tokens work in SQL Query, MDX Query, and Cube Value. They do **not** work in Text Value or Text List parameters.

- A referenced parameter with many values multiplies the work: N referenced values means the query runs N times and the results concatenate.

Related topics

- [Parameterize a report](#)
- [Parameters reference](#)
- [Worksheet output options](#)

Destinations

DIST-03

reference

report author

administrator

Destinations on the Distribution step are the delivery targets for a report's generated files. Each target type is a **distributor**. A distributor is configured in two places: a **global connection** (Settings, Integrations, Distributors, holding credentials, defaults, and a Test Connection button) and a **per-job destination** added on the Distribution step (path, folder, or recipient fields, most of them tokenizable, see [Name and organize output files](#)).

A job can target **multiple destinations at once**, and each destination delivers only the output formats you route to it. The Destinations area shows **no tiles until at least one distributor is configured** in Integrations; ask your administrator if the tile you need is missing.

The destinations

Destination	Writes output?	Per-job fields	Key behavior
Email	Yes	Recipients and message (address book plus contact tokens); From override if allowed; packaging options	Delivers files as attachments or a message. Supports Zip, combined messages, and message templates.
Network Folder	Yes	Network Location, a local or UNC path (tokenizable)	On-premises file-share delivery. Runs fully offline. Auto-creates missing folders.
SharePoint	Yes	Location URL (tokenizable)	Delivers to a SharePoint library folder via Microsoft 365 Graph. Auto-creates folders.
Box	Yes	Send As User, Folder Path	Server-to-server auth; folder auto-create; new version on a name clash.
OneDrive	Yes	Folder path; optional Share With plus permission	Writes to the delegated service account's own drive; replaces on re-run; optional non-fatal recipient share.
Google Drive	Yes	Folder, Drive Id, or email (contact fields)	Uses the Google Drive Provider connection. Excel output auto-converts to a Google Sheet on delivery.
Slack	Yes	Channel (contact token: Slack Channel)	Posts the output to a channel. See setup note below.
Teams	Yes	Chats or channels target (contact token or semicolon-delimited) plus message	Teams Direct (Graph) only in v6, see setup note. Email addresses go to chats; channel names go to channels, sent as the authenticated M365 user.
SFTP	Yes	File location / output path	Delivers to a secure FTP folder. See setup note below.
IBM Planning Analytics	Yes	A target PA connection plus an Application Folder path (forward-slash delimited, tokenizable)	A distributor that writes the job's output files back into a PA / TM1 Application Folder so PA users open the same file inside their model. Auto-creates missing folders.

IBM Planning Analytics is a destination, not just a source. The PA distributor *writes* output files into an Application Folder. This is the opposite direction from the PA *report provider*, which *reads* source reports from application folders. Same "Application Folders", opposite direction, do not conflate them.

Box is browse and read-only as a source, and a full write destination. As a source report provider Box is used to load templates; as a distributor it writes output. (There is no separate limitation to note for Box delivery beyond the folder and versioning behavior above.)

Per-destination setup notes

These are the non-obvious prerequisites that make a distributor work. Most are configured once, by an administrator, on the global connection.

Email (Microsoft 365 Graph): a non-default From needs a "Send as" grant

With the Microsoft 365 Graph transport, sending "as" any address other than the authenticating account's own mailbox requires that account to hold **Send as** on the target mailbox. In the Exchange Admin Center, open Recipients, Mailboxes, the target mailbox, Delegation, **Send as**, and add the authenticating account. Without it, an override-From send fails.

Network Folder: the Bootstrapper service account needs write access

The Network Folder connection has **no credential field**, only a Test Folder. The destination path is written by the **Reportworq Bootstrapper service account**, so that account (not the interactive user) must have write access to the target folder. Mapped drive letters are invisible to the service context; use a local path or a UNC path such as `\\MyServer\MyFolder`.

SharePoint: use the case-sensitive `.../sites/<site>/...` URL

The Location must be the canonical, **case-sensitive** form `https://<tenant>.sharepoint.com/sites/<site>/<folder1>/<folder2>`, **not** the URL from the browser address bar (which is encoded or redirected and will not resolve). The authenticating account needs rights to **create, delete, and update** files in the target library.

Slack: four OAuth scopes plus the bot in each channel

The Slack app needs four Bot Token OAuth scopes: `chat:write` and `files:write` (to post reports), plus `channels:read` and `groups:read` (to resolve a channel by name). Separately, the **bot must be added to each target channel** (in the channel, Integrations, Add an App), or posts fail. These are the two reasons a Slack distributor appears to "do nothing".

Teams: Direct (Graph) only, there is no Bot mode in v6

Only **Teams Direct (Graph)** is wired. The `Distribute` path splits the target list into email addresses (sent as chats) and channel names (sent as channel messages), as the authenticated user. A "Teams Bot" mode is **not shipped** in v6, do not plan around a Bot send path.

SFTP: provision the server account first

Reportworq needs a server-side, SFTP-enabled account with **full permissions on a dedicated folder**, so it can create the per-job subfolders it delivers into. The Host Address is the server's IP address.

How delivery behaves

Aspect	Behavior
Availability	No destination tiles until a distributor is configured and enabled in Integrations.
Tokenized paths	Path, folder, and recipient fields resolve parameter and variable tokens at run time.
Test Connection	Verifies credentials and reachability. It does not always verify per-job write access (notably Box and OneDrive).
Re-run	Box and OneDrive version, update, or replace a same-named file in place rather than duplicating.
Send gate	A destination delivers only when the run is not Hold for Delivery and is not suppressed by an untriggered exception check, and only the formats routed to it.
Failure isolation	Each destination runs independently. One broken destination (for example an unreachable SharePoint) is logged as a warning; the others still deliver.

Use cases

- **Email a monthly P&L PDF to controllers** with address-book recipients.

- **Publish a board deck to SharePoint**, delivered to a library folder built from a region token.
- **Archive a timestamped Excel to a network folder** per cost center.
- **Notify a finance channel** with a Slack or Teams post of the run's output.
- **Hand off a CSV extract to a partner over SFTP**.
- **Round-trip a report into Planning Analytics**, delivered back into a PA Application Folder so PA users open the same file in their model.

Notes and limits

- Test Connection may not confirm per-job write access.
- OneDrive writes only to the delegated service account's own drive.
- Teams delivers only via the Direct (Graph) path in v6.

Going deeper. To send one combined email per recipient instead of many separate messages, and to zip or collate the outputs, see [Combine emails, collate, and zip](#).

IBM Planning Analytics functions

reference

report author

Reportworq works with IBM Planning Analytics (TM1) two ways in a source workbook: the `RWMDX` importer, which runs an MDX query and writes the result into a range, and the **native TM1 worksheet functions** (`DBRW` , `DBS` , `SUBNM` , `DIMIX` , `VIEW` , `TM1RPT*` , and the element/dimension helpers) that Planning Analytics authors already know from Perspectives and Planning Analytics for Excel. Reportworq ships and evaluates this native library, so an existing TM1-authored workbook keeps working as a Reportworq report.

Before you use these functions

- A configured **IBM Planning Analytics / TM1** connection must exist for the workbook. An admin sets this up in the integrations hub; see [Connect IBM Planning Analytics](#).
- The functions are evaluated by the Reportworq engine at report-generation time; they do not compute in desktop Excel unless the corresponding IBM add-in is installed there.

RW.PlanningAnalytics.MDX

Runs a TM1 MDX query and writes the result set into an Excel range. Alias: `RWMDX` .

Signature

```
=RW.PlanningAnalytics.MDX(targetRange, connectionName, query)
```

#	Argument	Type	Meaning
1	<code>targetRange</code>	Range	Top-left output range for the MDX result.
2	<code>connectionName</code>	String	The TM1 connection to use.
3	<code>query</code>	String	The MDX query text.

All three are required.

Behavior

- Reportworq calls the configured TM1 REST provider and requests a cellset.
- **Two or more axes writes a crosstab** with row and column labels. **One axis writes a list-style output.**

- On success the formula cell is set to the query text that produced the data, so the delivered workbook shows exactly what was run.
- Reportworq attempts to destroy the TM1 cellset after writing the output.
- Leave enough space below and to the right of `targetRange` for the headers and the data values.
- Formula parsing succeeds before the MDX is ever sent, so a bad query surfaces at run time rather than at author time. Validate your MDX in a query tool first.

Examples

```
=RW.PlanningAnalytics.MDX(A1, "TM1Server1", "SELECT NON EMPTY [Time].Children ON ROWS,
{[Measures].[Revenue]} ON COLUMNS FROM [Sales]")
=RW.PlanningAnalytics.MDX(B5, "FinanceTM1", "{[Period].[2026]} ON COLUMNS FROM [Plan]")
```

When an argument is wrong

Situation	Result
Invalid <code>targetRange</code>	Formula error before the query runs.
Empty query text	Formula error.
The MDX returns an unexpected cell count for its axes	Report generation fails for that formula.

Available since around v5.0.0.80. Version availability is approximate.

Splitting a long MDX query across cells

An MDX (or SQL) query that is too long for one Excel cell can exceed Excel's roughly 8,192-character formula limit. Break it across several cells and concatenate them, for example `=A$1&A$2&A$3`. Two traps to avoid:

- **Every fragment cell must start with `=`**. A fragment cell that does not begin with `=` embeds its leading quote character literally into the query string, and the query then fails silently. Make each fragment its own formula.
- **Escape quotes around attribute values** in MDX, for example `&"&"&$A$1&"&"&`. Validate the concatenated output in a query tool before you deploy the workbook.

You can use `IF(...)` across the fragments to switch between queries.

Native TM1 worksheet functions

These are the classic TM1/Planning Analytics functions. Reportworq recognizes and evaluates them against the workbook's TM1 connection during report generation.

Function	What it does	When to use
DBR , DBRW	Retrieve a single cube cell value for a given element tuple. DBRW is the "register" variant used in most live grids.	The everyday way to pull one TM1 number into a cell.
DBRA	Retrieve an element attribute value.	To show an element's attribute (for example a description) beside its data.
DBS , DBSW , DBSS	Send (write) a value back to a cube cell. DBSS sends a string.	Write-back grids that push values into TM1.
SUBNM	Return an element name from a subset by index, and drive a picklist.	Parameter and picker cells that select the element a grid slices by.
VIEW	Reference a named cube view for retrieval.	To bind a block of DBRW formulas to a defined view.
TM1RPT*	The Active Form family that expands a dynamic report region.	Active Form reports that grow and shrink with the underlying subset.
DIMIX	Return the index of an element within a dimension.	Lookups and validation against dimension order.
DIMSIZ	Return the number of elements in a dimension.	Sizing loops and dynamic ranges.
DNEXT	Return the next element in a dimension.	Walking a dimension in order.
DNLEV	Return the number of hierarchy levels in a dimension.	Level-aware layout logic.
DTYPE	Return an element's type (numeric, string, consolidated).	Branch formatting or logic on element type.
ELCOMP , ELCOMP	Return a component (child) of a consolidated element, by name or by index.	Walk a consolidation's children.
ELISCOMP	Test whether one element is a component of another.	Membership checks in a hierarchy.
ELISPAR	Test whether one element is a parent of another.	Parentage checks in a hierarchy.
ELLEV	Return the level of an element.	Level-based indentation and formatting.
ELPAR , ELPARN	Return a parent of an element, by name or by index.	Roll-up navigation.

Function	What it does	When to use
ELSLEN	Return the number of elements in a subset.	Sizing a subset-driven range.
ELWEIGHT	Return the weight of a component in its consolidation.	Weighted roll-up calculations.

Universal Report functions

These belong to the same native library and are supported in **full (dynamic) Universal Reports**. They are **not** supported in static Universal Reports.

Function	What it does	When to use
TM1VAL	A hierarchy-aware retrieval, the complement to DBR .	Universal Reports that need hierarchy-aware cell values.
DefineCalc , DEFINECALC	Declare a user-defined calculation.	Custom calculated members inside a Universal Report.

Notes and limits

- The native functions are the standard TM1 syntax; refer to your IBM Planning Analytics documentation for exact argument lists.
- If a cell shows #VALUE! from a Planning Analytics for Excel workbook, the connection's Calc Mode can render invalid elements that way and can flip SUBNM / DIMNM public-versus-private subset resolution. See [Connect IBM Planning Analytics](#).
- Disable other datasources pointed at the same TM1/PA database before certain operations, concurrent connections to one database can conflict. This is a connection-level concern, covered on the connector page.

Related pages

- [Functions overview: which one do I use](#)
- [Relational database functions](#) for the SQL equivalents (RWSQL and the split-long-query rule).
- [General report and layout functions](#) for shaping the imported grid.
- [Connect IBM Planning Analytics](#) for the connection and authentication setup.

Review a run's performance

[how-to](#)[report author](#)[administrator](#)

Every run measures itself: how long it took, how much memory and CPU it used, and which formula families and calculation services accounted for the time. Nothing has to be turned on, and no setting controls it. Four places read that measurement, and this page covers the first two:

- **Run History** compares each run against the same report's own recent runs, so a slowdown is visible in the list.
- The **Performance** tab on a run's diagnostics breaks one run down until the cost is attributable to a specific output, workbook, processing phase, or calculation engine.
- The **Compare** tab searches the report's whole history for the point where its typical run time changed, see [Find when a report got slower](#).
- **Settings ▶ Performance** ranks every report against its own baseline and pivots the instance by release and by node, see [Compare performance across reports](#).

When to use it. When a pack that used to finish in three minutes now takes twenty, when you need to know which workbook in a large burst is the expensive one, or when you suspect a formula is being recalculated far more often than it needs to be. Also for capacity planning: peak memory and maximum CPU per run tell you what an execution host has to hold.

Before you start

- Open the report's **Run History**: in the workspace browser, open the report row's ... menu and select **Run history**. See [Track activity and run history](#).
- The three performance columns are visible to anyone who can see the run list. The **Performance** tab is not: it sits behind **View diagnostics**, which a view-only **User** account does not have. Authors and administrators reach it.
- Only runs executed on a build that records performance telemetry carry the data. Earlier runs say so rather than reporting zero.

Compare a run against its own history

The run list carries three performance columns.

Column	What it shows
Elapsed vs baseline	How long the run took, with a change indicator beside it. The indicator's tooltip names the comparison, for example a baseline of 3m 40s taken as the median of the previous 12 comparable runs.
Trend	A small chart of the report's recent elapsed times with the baseline drawn in, so a step change reads as a shape rather than as a number. A run with no telemetry breaks the line instead of dropping it to zero.
Peak mem	The highest working set the run reached, in MB or GB.

Workspace > Timberline > Financial Reports > Sales Rep Scorecard

Sales Rep Scorecard

Run history

Run now

Run history.

Every run of this report - newest first. Each run shows the outputs it produced, their status, the parameters used, and the formats delivered. Open a run to inspect outputs, downloads, and how each was distributed.

Runs are kept for 30 days (system default); older runs are removed automatically. Pin a run to keep it forever.

All 151

Success 151

	STATUS	STARTED	ELAPSED VS BASELINE	TREND	PEAK MEM	OUTPUTS	FORMATS	DELIVERY	
☐	Success	Aug 3, 2026 · 01:32	19.8s ~ as usual		952 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:28	24.9s ▲ +39%		982 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:24	17.4s ~ as usual		984 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:20	17.0s ~ as usual		965 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:16	18.8s ~ as usual		986 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:12	18.9s ~ as usual		977 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:08	19.2s ~ as usual		1000 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:04	22.6s ▲ +31%		950 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 01:00	20.3s ▲ +10%		937 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 00:56	17.7s ~ as usual		977 MB	13 variants · 13 files		✓ Email	> ...
☐	Success	Aug 3, 2026 · 00:52	16.7s ~ as usual		971 MB	13 variants · 13 files		✓ Email	> ...

4 of 151 runs recorded no performance telemetry — those columns read “—” rather than zero. 4 runs are held out of the baseline (failed, cancelled, or no telemetry), so a broken run can’t drag the comparison.

Five rules govern the comparison, and they explain why a delta is sometimes absent:

- The baseline is the **median of up to 12 earlier comparable runs** of the same report. The median is deliberate: one pathological run, a datasource timeout or a ten-times-larger burst, would drag an average far enough to hide every later regression.
- A report needs at least **3** earlier comparable runs before any comparison is offered. Below that, the column reports no comparison rather than measuring a run against a single predecessor.
- A difference within **15 percent** counts as noise and is not flagged.
- A run is **never part of its own baseline**.
- **Failed, canceled, and no-telemetry runs are held out** of the baseline, and they get no delta of their own. A job that died after 30 milliseconds would

otherwise be reported as the fastest run in the list.

Below the list, a note states how many runs recorded no telemetry and how many are held out of the baseline. A column full of dashes therefore reads as missing history rather than as a broken screen.

Two different elapsed figures. Where telemetry exists, the elapsed time is measured from execution start. Where it does not, the column falls back to the wall-clock duration on the run record, and the value's tooltip says which one you are looking at. The two are never mixed inside one trend.

Open the Performance tab for one run

1. In **Run History**, select the run you want to inspect.
2. Select **View diagnostics**.
3. Select **Performance**. The tab is present on every run, alongside **Diagnostics** and **Compare**. **AI diagnostics** appears next to them only for a run that failed or completed with warnings.

This run took 20.8s, and TM1 accounted for 34% of it.

Whole run

ELAPSED

20.8s

at 16,850 formulas

OUTPUTS

13 × 13

varieties × workbooks

RECALCULATION

8.5s

143 passes

PEAK MEMORY

974 MB

avg 719 MB

MAX CPU

19.7%

24 logical processors

DOMINANT ENGINE

TM1

7.1s total

Outputs in this run

Click a row to drill in

#1 E0005 2026

13.2s

21.7%

#1 E0007 2026

12.2s

20.1%

#1 E0006 2026

12.0s

19.7%

#1 E0008 2026

3.4s

5.6%

#1 E0010 2026

3.0s

4.9%

#1 E0011 2026

2.8s

4.7%

#1 E0015 2026

2.6s

4.2%

#1 E0009 2026

2.3s

3.8%

#1 E0017 2026

2.2s

3.6%

#1 E0012 2026

2.2s

3.6%

#1 E0013 2026

2.0s

3.2%

#1 E0016 2026

1.8s

2.9%

#1 E0014 2026

1.1s

1.9%

Engines

Rolled up across the run

ENGINE	CALLS	TOTAL	AVG	RECALC	PASSES
TM1 service	13	7.1s	546ms	once	
DBRW formula	308,112	5.0s	0ms	11 of 11	
DBR formula	51,912	761ms	0ms	11 of 11	
RWSort service	13	573ms	44ms	once	
TM1RPTELISCONSOLIDATED formula	4,016	362ms	0ms	11 of 11	
TM1RPTROW formula	286	179ms	1ms	11 of 11	
RWResize service	13	51ms	4ms	once	
TM1RPTVIEW formula	286	44ms	0ms	11 of 11	
RW_TOOLS_SORT formula	286	27ms	0ms	11 of 11	
SQL service	13	25ms	2ms	once	
Excel Built-in formula	51,152	15ms	0ms	11 of 11	
TM1RPTTITLE formula	572	14ms	0ms	11 of 11	
RW_TOOLS_RESIZE formula	286	6ms	0ms	11 of 11	
Relational service	13	3ms	0ms	once	
Adaptive service	13	2ms	0ms	once	
RWSuppress service	13	2ms	0ms	once	
RWAI service	13	1ms	0ms	once	
AdaptiveCalculationFormulasService service	13	1ms	0ms	once	
RWFormula service	13	1ms	0ms	once	
RWRemoveFormula service	13	1ms	0ms	once	
RWBlankRow service	13	0ms	0ms	once	
RWExternal service	13	0ms	0ms	once	
RWPAGEBREAK service	13	0ms	0ms	once	
RWGroup service	13	0ms	0ms	once	

Recalculation passes are counted against the busiest workbook in scope.

A run with no telemetry shows a short explanation in place of the breakdown. Two cases produce it: the run finished before performance telemetry existed, or it was a held-report release, which deliberately does not overwrite the original run's measurements.

Read the breakdown

The tab leads with one sentence naming the dominant cost, for example that the run took 4m 12s and one engine accounted for 63 percent of it. Where no single engine dominates, it says the time was spread across several engines instead.

Six figures follow.

Figure	How to read it
Elapsed	Total time, with the volume it covered underneath: data points where the run recorded them, otherwise the formula count. Volume is what makes "twice as slow" and "twice as slow for twice the data" different findings.
Outputs	The shape of the run, as generated outputs by workbooks.
Recalculation	Time spent recalculating, and how many passes it took.
Peak memory	Highest working set, with the average beside it. A long run's sample timeline is thinned, and the figure says when that happened.
Max CPU	Highest process CPU, with the host's logical processor count. A run too short to sample twice reports that CPU was never measured, not zero.
Dominant engine	The costliest formula family or calculation service, and its total.

Walk from the run down to a phase

Below the figures, one list ranks the current scope's contents by cost, largest first, with a bar showing the same proportions. Select a row to go one level deeper, and use the path above the list to come back up. The walk is always the same three levels:

1. **Whole run**, listing the outputs it generated.
2. **One output**, listing the workbooks inside it.
3. **One workbook**, listing its processing phases. Phases are the last level.

At workbook level, two tails are listed separately from the ranked phases: phases that **ran but cost nothing measurable**, and phases that **never ran at all**. Keeping them apart matters, because "this workbook has no charts" and "charts were free" are different facts about the same run.

Read the engine table

The **Engines** table lists one row per formula family (for example the Planning Analytics or SQL retrieval functions, with all built-in Excel functions folded into a single row) and per calculation service, with **Calls**, **Total**, **Avg**, and **Recalc passes**.

Recalc passes is the column to read. It says `once` for an engine that ran a single time, or `N of M` for one that re-ran during recalculation. An engine reading `5 of 5` on a five-pass workbook ran on every pass, and Reportworq highlights that row when the pass count is high enough for it to be expensive. That pattern is worth acting on: a data retrieval repeated on every recalculation pass is usually a formula that can be restructured, not an unavoidable cost. Engines that ran at no measurable cost are counted in a note under the table rather than listed.

Above workbook level the pass counts are folded together from the workbooks in scope, and the table says so, so `5 of 5` there means "in the recalculation loop the whole way" rather than a run-wide total.

Notes and limits

- **Runs that predate telemetry never gain it.** The measurement is written when the run executes, so history from an earlier build stays empty. Look at the runs from the current build onward.
- **A failed run still records its performance,** deliberately, since a failure is often exactly the run whose breakdown you want.
- **Releasing a held report writes no new measurement.** The release re-opens the original run's record, and overwriting it with a distribution-only measurement would destroy the history this feature exists to build.
- **Performance data ages out with the run.** It is stored with the run's other diagnostics and is removed when retention removes the run. **Pin** a run to keep it, for example a period-close run you expect to compare against. See [Output retention](#).
- **This is not alerting.** There are no thresholds to set and no notifications. The baseline rule is fixed as described.
- A view-only **User** never reaches the Performance tab, because **View diagnostics** is not available to that role.

Going deeper. For what to change once you know where the time went, see [Manage and improve performance](#). To find the point in this report's history where its run time changed, rather than judging one run, see [Find when a report got slower](#). Administrators can ask the same question across every report at once, and pivot by product release or execution node, in [Compare performance across reports](#). To read a failing run's log, attachments, and HTTP trace instead, see [When a job fails](#). To change how long runs and their diagnostics are kept, see [Output retention](#).

Profile and stress-test a campaign

[how-to](#) [report author](#) [administrator](#)

Campaign profiling answers two capacity questions before a campaign goes live: *how much server memory does a form cost*, and *what will contributors actually experience when many of them fill forms at the same time*. It is a diagnostics screen on the **Campaign Dashboard**, not part of the contributor experience.

The screen has three tabs, each a self-contained tool:

Tab	Answers	How it works
Memory profiling	How much memory does this form cost the server?	Opens the form in an isolated worker on the server and measures its footprint. Read-only.
Stress testing	Will the deployment hold up under real load?	Builds plan files for the rw-stress-runner , a separate tool that drives real browsers as virtual contributors.
Analysis	What did contributors experience?	Reads the result files the runner produced and reports the timings in plain language.

The worked example throughout is a **Q3 budget round** with two forms, *North America Budget* and *EMEA Budget*.

Before you start

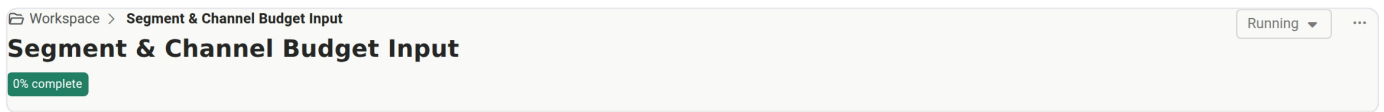
- The campaign has generated forms. Memory profiling and stress plans both read the generated form workbooks.
- For a stress test, set up a **dedicated stress-test user** and a **disposable copy of the campaign** (see [Recommended setup](#)).
- To run a stress test you need one or more **Windows** load-generator machines with the **.NET 10 runtime** installed. The load generators are Windows regardless of what the server runs.

Open Campaign profiling

1. Open the campaign's **Campaign Dashboard**.
2. On the dashboard toolbar, select **Campaign profiling**.

The screen opens on the **Memory profiling** tab. It profiles the forms you had selected in the dashboard, or every form in the campaign when nothing is selected. The breadcrumb reads **Workspace ► Campaign ► Profiling**; select **Close** to return to the dashboard.

To jump straight to memory profiling for a single form, open that form's **Actions** menu and select **Profile Memory**. The screen opens on the Memory profiling tab with only that form selected; you can add more.

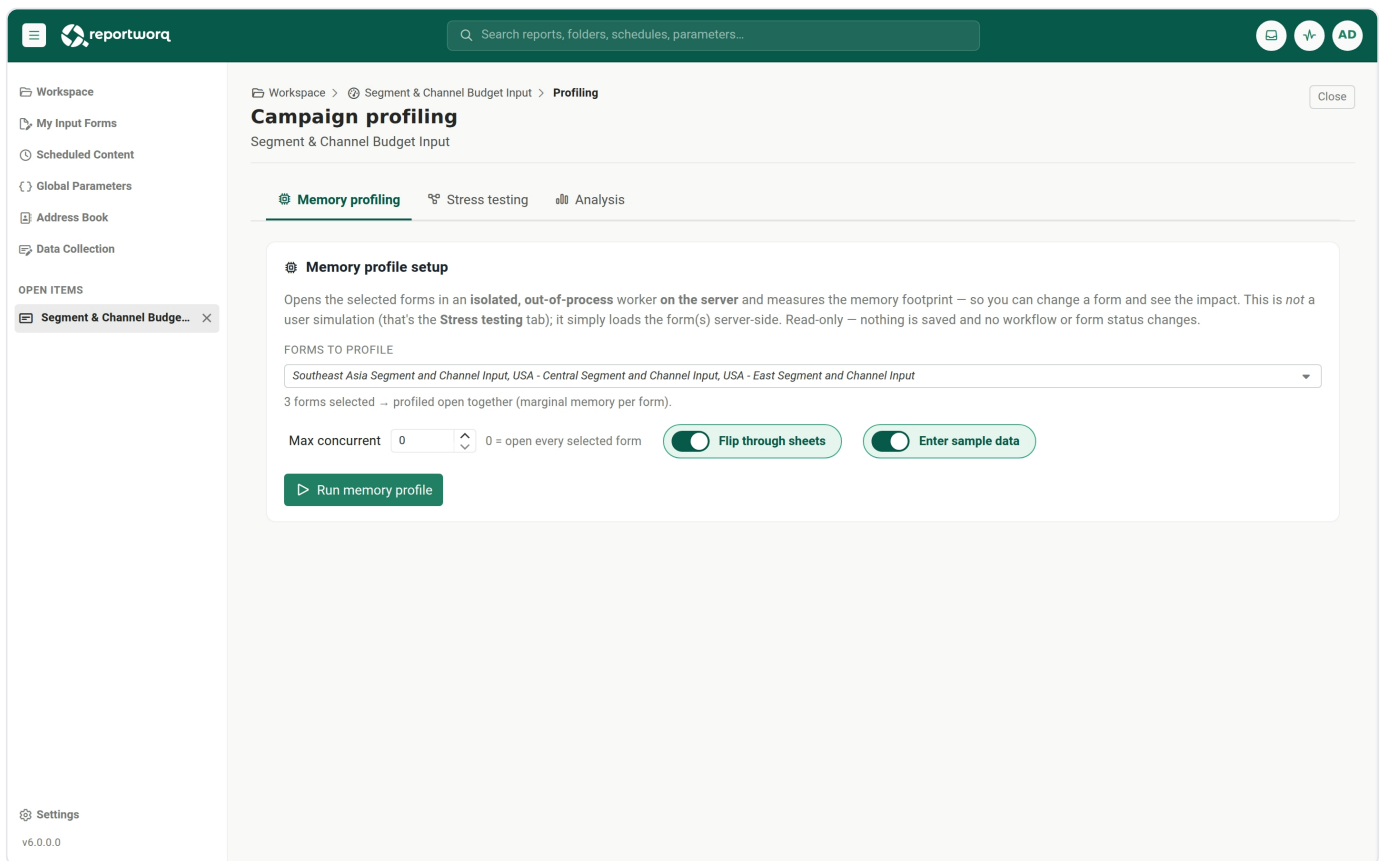


Memory profiling

Memory profiling opens a form in an **isolated, out-of-process worker on the server** and measures its memory footprint, so you can change a form and see the effect on cost. It is **not** a user simulation; that is the Stress testing tab. It is read-only: nothing is saved, and no workflow or form status changes.

Run a profile

1. Under **Forms to profile**, select one or more forms from the dropdown. The number you pick sets the mode:
 - **One form** gives a detailed per-form footprint measured over several cycles.
 - **Several forms** profiles them all open at once, to measure the marginal cost of each additional form.
2. Set the run options:
 - For a single form, set **Cycles** (1 to 10, default 3). Each cycle opens, measures, and closes the form; the first cycle is a warm-up and is excluded from the averages.
 - For several forms, set **Max concurrent** (0 opens every selected form).
 - **Flip through sheets** and **Enter sample data** exercise the form the way a contributor would. Both are on by default; leaving them on gives a more realistic footprint.
3. Select **Run memory profile**. The worker profiles on the server while the button shows progress.
4. To keep the numbers, select **Download CSV** or **Download JSON**.



Read a single-form profile

The **Per-form memory footprint** card leads with the numbers that matter:

- **Peak per form (avg)** is the working memory one open form costs at its peak, averaged across cycles, with the observed range.
- **Steady (just opened)** is the cost right after the form opens, before any interaction.
- **Bytes per cell (at peak)** relates cost to form size, alongside the cell and input-cell counts.
- **Held after close** is memory still held after the form closes, a retention or leak signal.

Below the stats, an estimate reads **~N of this form open at once on the profiling machine**, derived from the free memory on the profiling machine (using 80% of it, divided by the average peak per form). Treat it as **indicative**: real capacity depends on the server and the worker pool. The **Form & run facts** and **Per-cycle detail** cards give the underlying sizing and the raw per-cycle measurements.

Read a multi-form profile

When several forms are profiled together, the **Forms open together** card reports **Forms held open**, the **Marginal cost per form** (the true cost of each additional

concurrent form, after the first), the **First form** cost (which includes one-time warm-up), and the **Machine memory peak**. The **Ramp** table shows memory climbing as each form opens.

If a form cannot be opened, most often because its generated workbook has not been created yet or its saved copy is empty, it is listed under **Skipped**, with the reason, and the rest are profiled normally.

Stress testing

Stress testing measures the **live contributor experience** under load. The screen here builds the plan files; the load itself is generated by the **rw-stress-runner**, a separate executable that drives real browsers as virtual contributors. Each virtual contributor signs in, opens a form, reads it, and fills it in one cell at a time at human pace, the same path a real contributor takes, which is what actually stresses the server.

Stress tests change campaign data. The runner edits values and submits forms as real contributors, so it changes campaign state. Always run a stress test against a **disposable campaign**, reset the campaign before every run, and never point a stress test at a production campaign.

reportworq

Search reports, folders, schedules, parameters...

📧

🔔

AD

Workspace

My Input Forms

Scheduled Content

Global Parameters

Address Book

Data Collection

OPEN ITEMS

Segment & Channel Budge...

×

Workspace > Segment & Channel Budget Input > Profiling

Campaign profiling

Segment & Channel Budget Input

Memory profiling

Stress testing

Analysis

⚠️ Stress tests must run against a disposable campaign, always.

The runner edits data and submits forms as real contributors, so it changes campaign state. Reset the campaign before every run, and never point a stress test at a production campaign.

📎 Get the runner

Download the rw-stress-runner for Windows (the load generators are Windows machines, whatever the server runs) and copy it to each one. It needs the .NET 10 runtime installed. Then run `rw-stress-runner.exe install-browser` once and `rw-stress-runner.exe run --plan node-N.json`.

📎 Windows (x64) – needs .NET 10

101.6 MB

🔧 Plan setup

Buils partitioned plan files for the rw-stress-runner executable. It is the forms that are split across the nodes – each form runs on exactly one node, because a form allows only one editor at a time. **Best practice:** create a dedicated stress-test user and add it as a contributor on every form you're testing (forms can have several contributors, so this doesn't disturb the real assignees). The runner signs in as that one user for all the forms, so concurrency means concurrent forms and you manage a single login. (One user opening several forms at once is out of scope – it isn't realistic, so it isn't stress-tested.) Run each node file on a separate machine, at the same time.

Target URL

http://localhost:8300/

The deployment under test (defaults to this server's address).

Test-user name

the stress-test user's login (e.g. stresstest@your-co.com)

The dedicated stress-test user signed in on every form. Add it as a contributor to all the forms you're testing.

Test-user password

password for the stress-test user

Settings

v6.0.0.0

Recommended setup for a stress test

Prepare these three things before you build a plan:

- **A dedicated stress-test user.** Create one account and add it as a contributor on every form you want to test. Forms can have several contributors, so this does not disturb the real assignees. The runner signs in as this one user for all forms, so you manage a single login and concurrency means concurrent *forms*.
- **A disposable campaign.** Use a throwaway copy, never production content.
- **A reset before each run.** Because the runner writes data and submits, reset the campaign to a clean state before every run so results are comparable.

Get the runner

1. On the **Stress testing** tab, under **Get the runner**, download the **rw-stress-runner** for **Windows** and copy it to each load-generator machine.
2. On each machine, confirm the **.NET 10 runtime** is installed.
3. Install the browser once per machine:

```
rw-stress-runner.exe install-browser
```

If the download links are absent, the runner was not bundled in this build; an administrator can publish it from the `Reportworq.Contribution.StressRunner` project and place the kits where the app serves them.

Build the plan files

1. Fill in **Plan setup**:
 - **Target URL** is the deployment under test. It defaults to this server's address.
 - **Test-user name** and **Test-user password** are the dedicated stress-test user's login. The runner types these into the login form on each node.
 - **Nodes (machines)** is required. One plan file is produced per node. The **forms** are partitioned across the nodes; each form runs on exactly one node, because a form allows only one editor at a time.
 - **Concurrent forms / node** sets how many forms a single machine opens at once (default 10).
2. Set the **Editor pace** slider. It scales every human pause and the ramp-up together, so the same plan can finish in minutes (toward **Fast**) or run for an hour or more (toward **Slow & deliberate**). The **Estimated run time** updates live as you slide.
3. Optionally open **Advanced settings**. The defaults are tuned for a realistic run:

- **Cells to fill (%)** is the share of each form's input cells a user fills, as a range (default 50 to 75).
 - **Submitters (%)** is the share of users who submit rather than close (default 25, applies when **Submit form** is on).
 - The switches turn individual behaviors on or off: **Fill in cells**, **Spread (bulk)**, **Add comments**, **Recalculate**, **Change sheets**, and **Submit form**.
4. Select **Compute sizing & build plans**. Reportworq measures each form and partitions the work into node plans.
 5. Review the **Nodes** and **Form sizing** tables, then select **Download N plan file(s)**. The download is a `.zip` of one `node-N.json` per machine, plus a `README.txt` with the run steps.

The screenshot shows the Reportworq web application. The left sidebar contains a navigation menu with items like 'Workspace', 'My Input Forms', 'Scheduled Content', 'Global Parameters', 'Address Book', 'Data Collection', and 'OPEN ITEMS'. The main area is titled 'Campaign profiling' and has a sub-header 'Segment & Channel Budget Input'. Below this, there are tabs for 'Memory profiling', 'Stress testing' (which is active), and 'Analysis'. A yellow warning box states: 'Stress tests must run against a disposable campaign, always. The runner edits data and submits forms as real contributors, so it changes campaign state. Reset the campaign before every run, and never point a stress test at a production campaign.' Under the 'Get the runner' section, there is a download button for 'Windows (x64) - needs .NET 10' (101.6 MB). The 'Plan setup' section includes instructions and three input fields: 'Target URL' (with 'http://localhost:8300/' entered), 'Test-user name' (with 'the stress-test user's login (e.g. stresstest@your-co.com)' entered), and 'Test-user password' (with 'password for the stress-test user' entered).

Run the stress test

On each load-generator machine:

1. Copy the runner and that machine's single plan file to it.
2. Run the node plan:

```
rw-stress-runner.exe run --plan node-1.json
```

3. Launch every node at about the same time.

Each node writes `node-N.result.json`, `.csv`, and `.md`. Keep the `node-*.result.json` files; you upload them to the Analysis tab.

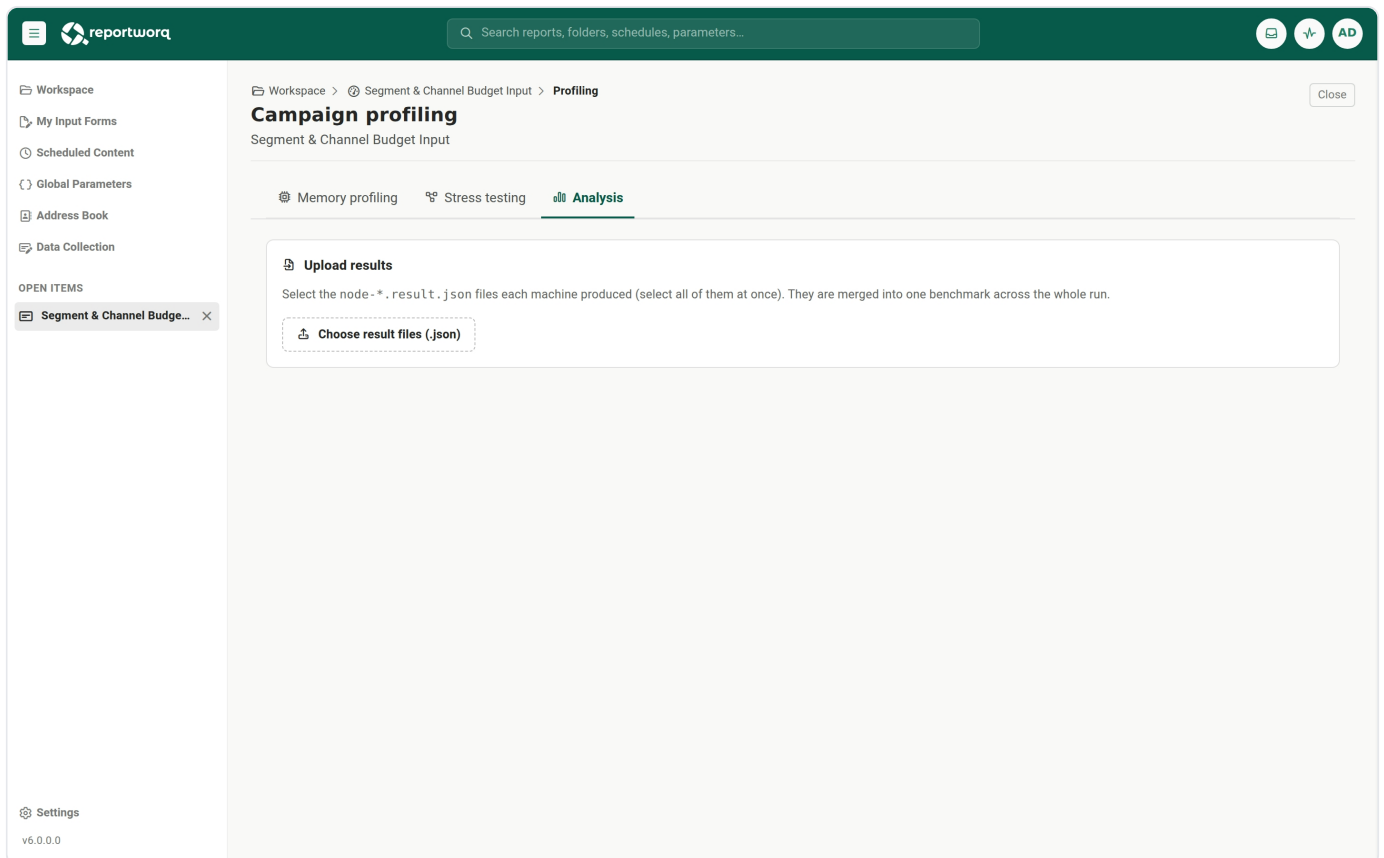
When you launch a run from a terminal, the runner asks two questions it does not read from the plan: whether to open the browsers so you can watch them (default is headless, which is best for a real run), and, for single sign-on, it opens a browser for you to sign in once. For a stress test that uses **SSO/OIDC** instead of a password, turn on the plan's SSO option and leave the passwords blank; each node prompts you to sign in when the run starts.

Analysis

The **Analysis** tab turns the node result files into one consolidated benchmark and describes, in plain language and in seconds, what contributors experienced.

Load the results

1. On the **Analysis** tab, select **Choose result files (.json)**.
2. Select every `node-*.result.json` file the run produced, from all machines, at once. They are merged into a single benchmark. Files that are not readable node results are skipped and counted.

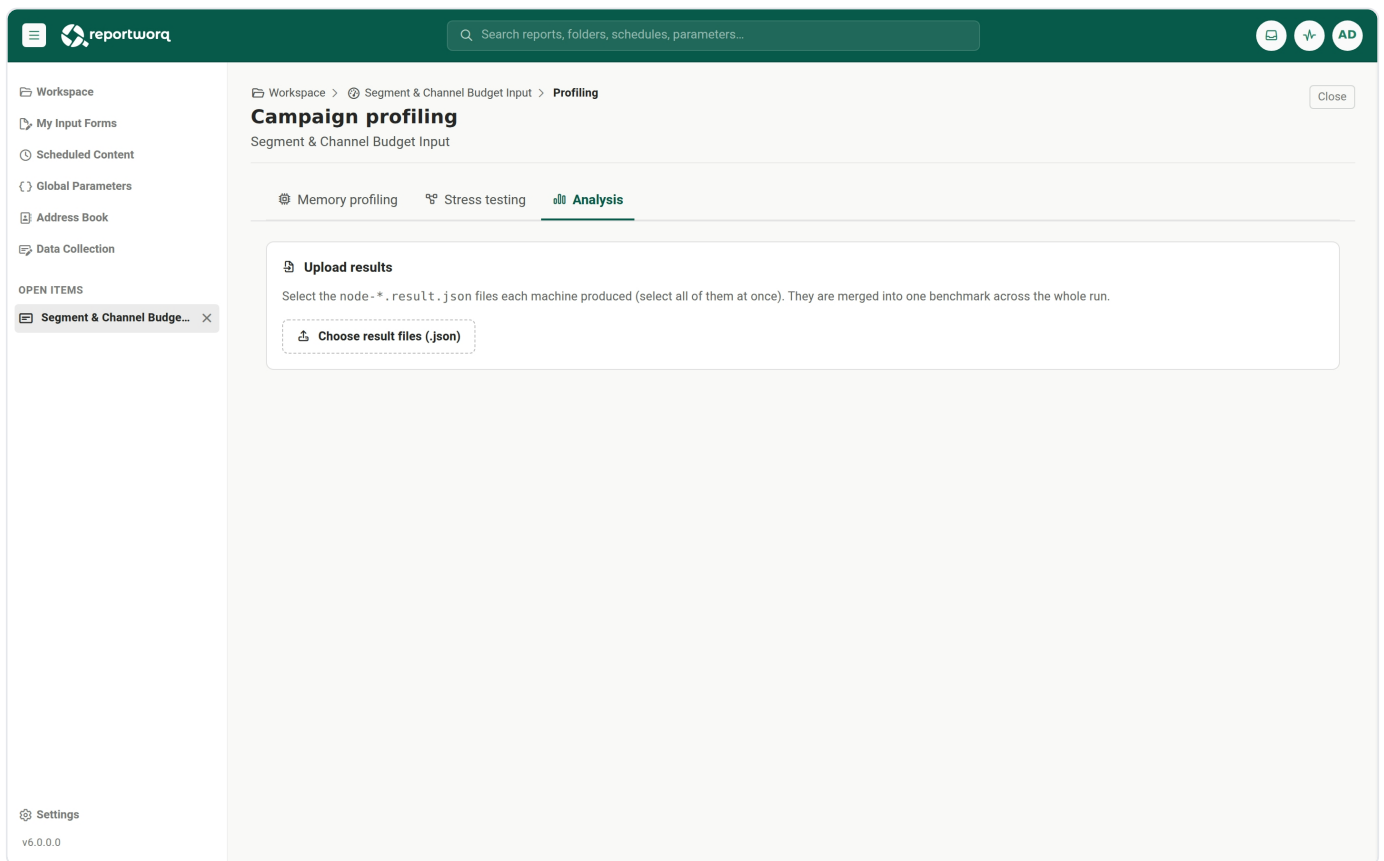


Read the report

- **Run summary** gives the headline scale: **Peak concurrent forms** (with the average and node count), the number of **Forms** and users, total **Actions** and errors, and the **Duration** with the machines that ran. If the run used more than one login, a note reminds you that best practice is a single dedicated test user, because concurrency is measured per *form*, not per person.
- **Findings** flags concerns automatically. A **slow tail** means most users were fine but the slowest few waited much longer; a **slowdown** means an action got slower as the run went on.
- **What users experienced** is the plain-language read. A short paragraph summarizes the run, and the table reports each action in seconds across three columns:
 - **Typical** is what half of users saw.
 - **Most users** is what 9 in 10 saw.
 - **Slowest seen** is the worst single measurement.

The rows cover the contributor actions: open the form, enter a value, spread, add a comment, recalculate, change worksheet, and submit. These are what a contributor's browser actually experienced end-to-end.

- **At a glance** charts the typical and near-worst time for each action so slow actions stand out. **Per-form** breaks the key timings down by form, next to each form's size.



For the full percentile breakdown and every raw measurement, select **Download details (Excel)**. The workbook has separate sheets for the summary, the full timings (min, mean, p50, p90, p95, p99, max), the per-form numbers, the findings, and every raw sample. **Download JSON** exports the consolidated benchmark.

Notes and limits

- Memory profiling is read-only and never changes campaign state. Stress testing changes campaign state and must run against a disposable campaign.
- Memory profiling and stress plans read a form's **generated** workbook. A form whose workbook has not been generated is skipped with a reason.
- The memory concurrency estimate reflects the **profiling machine**, not necessarily production hardware. Use it as a relative guide.
- The rw-stress-runner download is **Windows (x64)**; the load generators are Windows even when the server runs on another operating system.

Related topics

- [Scale and size a contribution campaign](#)
- [Author a contribution campaign](#)
- [Fill in and submit a form](#)
- [Contribution \(write-back\) overview](#)

Worksheet output options

AUTH-03

how-to

report author

Each worksheet you include in a report can carry its own **Output Options**, settings that control how that worksheet appears and behaves in the generated file. They are configured on the **Reports** step of the Job Editor, per included worksheet, alongside the worksheet's **Parameters** mapping.

These options apply to a single worksheet. Report-level packaging (collating and splitting) is set through [Job options](#).

Before you begin

- Include at least one worksheet on the [Reports step](#). Until an Output Options entry is added, the worksheet is generated with default behavior.

Set the options

On the **Reports** step, open a worksheet's **Output Options** panel and set any of the following.

Option	What it does
Sheet name	Sets the output worksheet's name, which can differ from the source tab. Use literal text plus parameter tokens, or read it from a calculated cell. The name also becomes the worksheet's PDF bookmark name.
Sheet tab color	Colors the worksheet tab in Excel output. Use a named color (<code>Red</code> , <code>Blue</code>), a hex code starting with <code>#</code> (for example <code>#333333</code>), or an integer ARGB value.
Output order	Sets the sequence in which this worksheet is rendered relative to others, as a manual integer or from a cell. Most ordering needs are met by collation in Job options .
PDF parent bookmark	Nests this worksheet under a parent in the PDF bookmark tree. The tree is three tiers: the root comes from Distribution Options, tier two defaults to the report name, and tier three is the worksheet name. Set a literal, tokens, or a calculated cell to group siblings under a custom parent.
Sheet suppression	Deletes the worksheet from the output when a cell or <code>RWSHEETSSUPPRESS</code> formula evaluates TRUE. See below.
Exception check	Gates whether the whole report is sent. See below.

When you set an option in the panel, it **wins over** the equivalent worksheet formula (`RW.Tools.SheetName` , `RW.Tools.SheetColor` , and `RW.Tools.SheetSuppress` ; the short aliases `RWSHEETNAME` , `RWSHEETCOLOR` , and `RWSHEETSUPPRESS` also work). The formula is used only when the panel option is left unset.

Page breaks are not a per-worksheet Output Option. Control them from within the worksheet with the `RWPAGEBREAK` function. See [General report and layout functions](#).

Suppress a worksheet conditionally

Sheet suppression **deletes** the tab from the output, it does not hide it, when its cell or `RWSHEETSUPPRESS` formula evaluates TRUE (a value of `1` or a boolean true). On FALSE the tab is kept.

A common pattern is a formula that keeps a tab only when it holds data, so recipients never receive an empty schedule. A worksheet is **also auto-suppressed when it carries a provider report that returns no data**.

When to use it: drop lookup or staging tabs so recipients see only presentation worksheets, or hide a schedule that has nothing to show this period.

Send only on an exception (the exception check)

The **exception check** turns a worksheet cell into a send gate for the **whole report**. The rule is an OR across every worksheet:

- If **no** worksheet defines an exception cell, the report always sends.
- If **one or more** worksheets define an exception cell, the report is delivered only when **at least one** of those cells, anywhere in the report, evaluates TRUE. If all of them are FALSE, delivery is suppressed.

When to use it: send a variance pack only to the managers who are actually over budget. Put an exception cell on a validation tab that turns TRUE when a threshold is breached, and recipients whose reports stay within tolerance receive nothing.

Notes and limits

- **Sheet tab color** applies to Excel output; **PDF parent bookmark** applies to PDF output. The options are format-relevant.
- Suppression removes the tab entirely. There is no keep-but-hidden state at this layer.

Related topics

- [Choose the source workbook and worksheets](#)
- [Job options](#)
- [Parameters reference](#)

Email messages and templates

DIST-03

how-to

report author

The email **subject** and **body** on the Distribution step are built in the same pill editor as file names, so you can drop **variable tokens** into the wording. That makes a message **dynamic**: it fills in the period, recipient, or figures at send time, so you are not editing the subject line and body by hand every week or quarter. Save a message you will reuse as a **message template**, and reload it across jobs and campaigns.

Before you begin

- You are on the **Distribution** step with the **Email** distributor added, and its message editor open.
- For contact tokens the job must burst to contacts or use contact mapping; for cell-value tokens the source workbook must contain the referenced cell or named range.

Write a dynamic subject and body

1. In the Email message editor, type the fixed wording of the subject and body.
2. Insert **variable tokens** where a value should fill in at send time. The **message body** offers the fullest set, the same tokens as a file name plus the richer content tokens that only make sense in a message:

Body token	Inserts	Notes
Cell Value	A source cell or named range	Resolves even if the worksheet is excluded from output . A render mode controls the form: plain value (optionally with cell formatting as a colored or bold span), HTML Table , Markdown Table , or Plain Text Table . Multi-cell plain values join with newlines.
Chart	A chart from the workbook	A Range option can pick the chart name from a cell, so the inserted chart is chosen at run time.
Image	An image from the workbook	Same run-time Range selection as Chart.
Screenshot	A rendered image of a worksheet range	Set the Range to capture.
AI Insight	The narrative produced by one of the job's AI prompts	Body only. See Add an AI-generated insight to a message .
Address Book (the contact token)	An attribute of the resolved recipient contact	Resolves only when a contact is associated with the iteration, that is, with burst sets or contact mapping.
Parameter / Global Variable / Timestamp	The same tokens as file names	Resolve per burst iteration and recipient.

3. To link to something external rather than attach it, use Markdown link syntax in the body, `[Link text](URL)`, and put a variable in the URL so the link is dynamic per recipient (for example a SharePoint link to that recipient's file).
4. Review the body, then save the job.

When to use dynamic tokens. The message goes out on a recurring schedule and its wording changes only by period, entity, or recipient. Tokenize those parts once and the weekly or quarterly edit disappears.

Recipients do not have to come from the address book

The **To**, **Cc**, **Bcc**, and **From** fields are token fields too, so a job can address its mail without any address-book contacts at all. Insert a **Cell Value** token and the recipient address is read from a cell or named range in the source workbook at send time. Because the resolved value is split on semicolons, one cell can hold several addresses, for example `ap@contoso.com; controller@contoso.com`.

That makes the workbook itself the recipient list, which is useful when the routing already lives in the report, a distribution tab maintained by the business, or a lookup that derives the owner of each cost center. The cell resolves even when its worksheet is excluded from the output, so the routing tab never has to be delivered.

Use the [Address Book](#) when the same people receive many different reports and you want their destinations maintained in one place. Use a Cell Value token when the list is short-lived, owned by the report, or already computed in the workbook.

Add an AI-generated insight to a message

An AI-generated insight is not an output format and not a file. It is model-written narrative that you **embed into the message body**, so the recipient reads a short written explanation above the numbers they have been sent.

This needs at least one **enabled AI provider connection**. Without one, the **AI Generated Insights** section does not appear on the Distribution step.

1. On the **Distribution** step, open the **AI Generated Insights** section and choose **Add** to create a named prompt. Prompts appear as a tab strip you can rename or delete; names must be unique.
2. Set the **AI Model** to one of your enabled connections. Choosing a model applies that provider's default token limit to the prompt.
3. Open **Settings** to set **Max Tokens** (minimum 10, maximum 1,000,000, default 2048) and, if wanted, **Include AI-Generated Footer**, which appends a note that the content was AI-generated and names the model.
4. Write the prompt. It accepts free text and Reportworq variable tokens, so it can reference the same parameter and cell values that drive the report, for example "Summarize the largest month-over-month variances for the department below." Images are not allowed in a prompt.
5. In the message body, insert the **AI Insight** token and select the prompt by name. At send time the prompt runs and its narrative is written into the message at that point.

When to use it. A burst sends each department its own P&L and each recipient should get a short commentary written for their numbers, rather than one generic paragraph reused across every department. Because the prompt reads the job's own variables, the narrative is personalized per burst iteration with no manual writing.

Notes. The AI Insight token is available in the **message body** only, not in the subject line, the address fields, or the output file name. An AI Insight cannot be nested inside another prompt. The generated text is also recorded in Run History's per-output **AI Insights** section, so you can read what each recipient was sent. For provider setup and

for the in-workbook alternative, see [AI-generated insights in reports](#).

Save and reuse a message template

A **message template** is a reusable subject, body, and category, stored per workspace, so a branded or standard message is written once and reloaded everywhere.

1. Compose the subject and body (with tokens and, if wanted, an embedded logo image for branding).
2. Choose **Save Template** and give it a name. Saving with an existing name **overwrites** that template with the current editor content.
3. On any other job or campaign, choose **Select a Template** to open the picker: templates are listed on the left with a live **preview** of the selected body on the right. Choosing one loads its subject and body into the editor.
4. **Rename** and **Delete** are available on the selected template in the picker.

When to use a template. The same, or nearly the same, message is sent across many jobs or campaigns, a branded cover email for every monthly pack, or a standard contributor notification. Write it once, reload it everywhere, and keep the wording consistent.

Use cases

- **Branded monthly pack email.** A logo header, standard greeting, and a dynamic link to the report, saved as a template and reloaded for every monthly P&L and forecast.
- **Dynamic subject line.** `Weekly P&L, [Store], [Timestamp]` fills in the store and run date, so the subject never needs a manual edit.
- **Figures in the body.** A Cell Value token drops the headline variance into the email text as a formatted value or a small table.
- **Narrated variance pack.** An AI Insight token adds a per-department commentary to each bursted email.
- **Routing held in the workbook.** A Cell Value token in the **To** field addresses each output from a distribution tab the business maintains, with no address-book contacts involved.

Notes and limits

- **Templates are workspace-scoped.** A template belongs to the workspace it was saved in.
- **Address Book tokens need a resolved contact.** They fill in only when the run associates a contact with the iteration, through burst sets or contact mapping.

- **The richer body tokens are body only.** Chart, Image, Screenshot, and AI Insight are offered in the message body, not in the subject or the address fields.
- **For a one-off message,** edit the body inline without saving a template.

Going deeper. To combine several outputs into a single templated email per recipient, pair templates with combining. See [Combine emails, collate, and zip](#).

Relational database functions

reference report author

The relational family reads data from, and writes data back to, SQL, OLE DB and ODBC sources. `RW.Relational.Export` imports a query result into the workbook; the update and upsert functions write values back during input submission.

Function	Alias	What it does
<code>RW.Relational.Export</code>	<code>RWSQL</code> , <code>RWRELATIONALEXPORT</code>	Runs a query and imports the result into a range or named table
<code>RW.Relational.Update</code>	<code>RWSQLUPDATE</code>	Updates the row matching a key column and value
<code>RW.Relational.Upsert</code>	<code>RWSQLUPSERT</code>	Same as Update, but inserts the row when the key is missing

Before you use these functions

- A configured **SQL, OLE DB or ODBC** connection must exist, referenced by name in the `connectionName` argument. An admin sets this up; see [Connect SQL, OLE DB and ODBC](#).
- `RW.Relational.Export` runs during **report generation**. The write-back functions run during **input submission**, not the output pass.
- The functions are inert in desktop Excel. They only run inside a Reportworq job or form submission.

RW.Relational.Export

Executes a SQL query and imports the result set into an Excel range or named table.

Signature

```
=RW.Relational.Export(targetRange, connectionName, query)
```

#	Argument	Type	Meaning
1	<code>targetRange</code>	Range or named table	Where the result is written. A table target imports into the table body.
2	<code>connectionName</code>	String	The configured SQL connection name.
3	<code>query</code>	String	The SQL query text.

All three are required.

Behavior

- A one-time report-generation formula. It runs **before** the layout functions, so `RW.Tools.Sort` , `RW.Tools.BlankRow` and `RW.Tools.Suppress` all operate on the fully expanded result.
- On success the formula cell is set to the query text that produced the data, rather than being cleared. That is how you can see, in the delivered workbook, exactly what was run.
- With a plain range target, the result is written starting at the supplied range. Leave enough room for the full result set.
- A multi-row range target is repopulated by a clear, delete, then repopulate cycle. It does not overwrite in place, and it does not disturb cells below or to the right of the range.

Targeting a named table

Use a table target when you want Excel tables, structured references, or downstream charts that grow with the result set.

A named table target must have "Show Header Row" enabled. The import writes into the table body below the header; without a header row it has nowhere to place the data.

For a copied formula, wrap a relative target in `INDIRECT("...")` so the reference does not drift as the formula is filled down. The modern alternative is to copy the sheets as a group, which rewrites cross-sheet references correctly.

Examples

```
=RW.Relational.Export(A1, "DefaultConnection", "SELECT * FROM Sales WHERE Year = 2026")
=RW.Relational.Export(SalesTable, "WarehouseDB", "SELECT CustomerID, Amount FROM Orders")
```

When an argument is wrong

Situation	Result
Invalid target range or table name	Formula error before the query runs.
Blank query text	Formula error.
The query returns more data than expected	The imported data expands into the target area.

RW.Relational.Update

Writes values back to SQL by updating the row that matches a key column and key value.

Signature

```
=RW.Relational.Update(connectionName, table, idColumnName, idColumnValue, columnName1, value1, [columnName2, value2], ...)
```

#	Argument	Type	Meaning
1	connectionName	String	The configured SQL connection.
2	table	String	The table to update.
3	idColumnName	String	The key column used in the match condition.
4	idColumnValue	Value	The key value to match.
5 and up	columnName , value	Repeating pairs	One or more columns to update.

At least six arguments are required, so that at least one column/value pair is present. Append further pairs in the same formula for additional columns.

Behavior

- Processed during input submission, not the report-output pass. The formula cell is cleared on success.
- Reportworq records tuple-log information for the connection, table, key and updated columns.
- The row must already exist. Unlike Upsert, Update never inserts.

Examples

```
=RW.Relational.Update("DefaultConnection", "Users", "UserID", 123, "Email", "new@email.com")
=RW.Relational.Update("CRM", "Accounts", "AccountID", A1, "Manager", B2, "Territory", C3, "Priority", "High")
```

When an argument is wrong

Situation	Result
Fewer than six arguments	Formula error during parsing.
Blank connection, table, key name, or key value	Formula error during parsing.
An odd number of trailing column/value arguments	The unmatched trailing item is silently ignored. Check your pairs; a mistyped column writes nothing and reports nothing.

RW.Relational.Upsert

Writes values back to SQL by updating an existing row, or inserting a new row when the key does not yet exist.

Signature

```
=RW.Relational.Upsert(connectionName, table, idColumnName, idColumnValue, columnName1, value1, [columnName2, value2], ...)
```

The arguments are identical to [RW.Relational.Update](#). Upsert shares the same parser and the same rules, including the six-argument minimum, the cleared formula cell, the tuple logging, and the silently ignored odd trailing argument. The one difference is that Reportworq enables insert-on-missing-key because the function name is `RW.Relational.Upsert`.

Use Upsert when the workbook may create brand-new rows as part of contribution processing, for example a forecast ID that does not exist yet. Use Update when the target row must already exist and you want a missing row to be a no-op rather than a new record.

Examples

```
=RW.Relational.Upsert("AnalyticsDB", "Forecasts", "ForecastID", "Q3_2026", "Revenue", 5000000, "Confidence", 0.95)
=RW.Relational.Upsert("CRM", "Accounts", "AccountID", A1, "Status", "Active", "Territory", B2)
```

Check that the target database and connection actually permit inserts before relying on this.

Splitting a long query across cells

A SQL query too long for one Excel cell can exceed Excel's roughly 8,192-character formula limit. Break it across several cells and concatenate them, for example `=A$1&$A$2&$A$3`.

- **Every fragment cell must start with `=`**. A fragment cell that does not begin with `=` embeds its leading quote character literally into the query string, and the query then fails silently. Make each fragment its own formula.
- For SQL, **add table aliases on ambiguous columns** so the assembled query is valid.
- Validate the concatenated output in a query tool before you deploy. You can use `IF(...)` across the fragments to switch between queries.

Use cases

- **SQL warehouse extract.** `RW.Relational.Export` imports orders into a named table so downstream charts grow with the result, then `RW.Tools.RemoveFormula` freezes the values before the file is emailed.
- **Planning write-back.** After a review, `RW.Relational.Upsert` writes updated forecast rows back to the planning database, inserting any new forecast IDs.
- **Corrections round.** A contribution form collects amended values, and `RW.Relational.Update` writes them to rows that are known to exist, so a typo in a key never silently creates a record.

Notes and limits

- Update requires the keyed row to already exist. Use Upsert when the row may be new.
- Keep write-back formulas out of the delivered output. They render as `#NAME` in a contribution form, which is why their rows and columns are hidden by design.
- All three functions date from around v5.0.0.60. Version availability is approximate.

Related pages

- [Functions overview: which one do I use](#)
- [IBM Planning Analytics functions for the MDX equivalent.](#)
- [General report and layout functions](#) for shaping the imported data.
- [Contribution functions](#) for the form controls these write-back formulas sit behind.
- [Connect SQL, OLE DB and ODBC](#) for the connection setup.

Find when a report got slower

[how-to](#)[report author](#)[administrator](#)

The **Elapsed vs baseline** column in Run History tells you whether the latest run was normal. That is the wrong question for a report that has been drifting for a fortnight, because every individual run looked fine against a baseline that was itself creeping upward.

The **Compare** tab asks the other question: across this report's whole history, is there a point where its typical run time changed, and when was it? It is not a comparison of two runs you pick. It searches for a single step, and it either finds one or says plainly that there is not one.

When to use it. When a report feels slower but no single run is flagged, when you want to line a slowdown up against an upgrade or a data-model change, or when you want to know whether one output of a large burst has drifted while the rest have not.

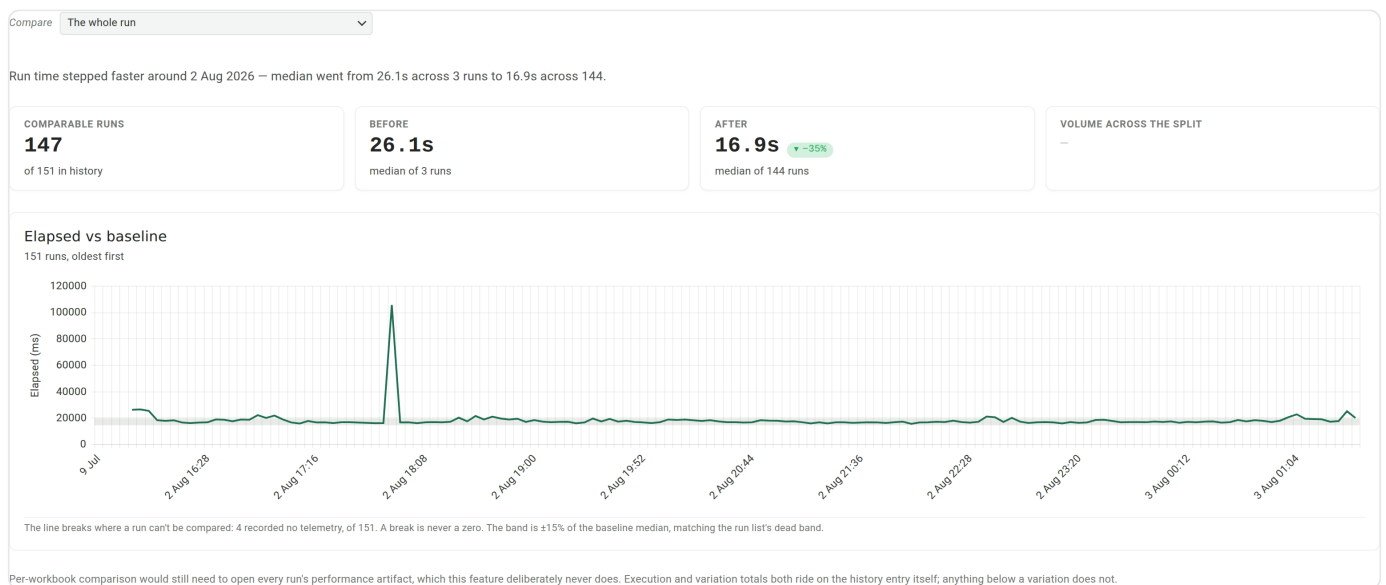
Before you start

- Open the report's **Run History**: in the workspace browser, open the report row's ... menu and select **Run history**. See [Track activity and run history](#).
- The tab sits behind **View diagnostics**, which a view-only **User** account does not have. Authors and administrators reach it.
- The search needs at least **six** comparable runs, three on each side of the point it is testing. A report with fewer says so rather than guessing.

Open the Compare tab

1. In **Run History**, select the run you want to start from.
2. Select **View diagnostics**.
3. Select **Compare**. The tab is present on every run, alongside **Diagnostics** and **Performance**. **AI diagnostics** appears next to them only for a run that failed or completed with warnings.

Which run you started from does not change the answer. The tab looks at the report's whole history, not at that one run.



Read the verdict

The tab leads with one sentence. It is the whole answer, and everything below it is the evidence.

Where a step is found, the sentence names the direction, the approximate date, and the medians on each side, for example that run time stepped slower around a given date and the median went from one figure across a number of runs to another figure across the rest. It then adds what the data can honestly say about the cause:

- **Volume.** If data volume moved by more than 15 percent across the same split, the sentence says so, because "it got slower" and "it got bigger" are different findings. If volume barely moved, it says that too, which is the stronger result: the report is not simply doing more work.
- **Release.** If the product version differs on either side of the point, the sentence names the change.
- **Node.** If the runs moved to a different execution instance, the sentence names that too.

A cause is named only when the data actually shows one. Where the runs on one side disagree with each other, nothing is named rather than a majority being presented as a fact.

Where no step is found, the sentence says the run time has stayed within its usual range across the whole window. That is a finding, not a failure.

Read the figures and the chart

Four figures sit under the verdict.

Figure	How to read it
Comparable runs	How many runs could take part, out of the runs in history. Failed, canceled, and unmeasured runs are excluded.
Before	The median run time on the earlier side of the step, and how many runs that is.
After	The median on the later side, with the change against Before beside it, and how many runs it covers.
Volume across the split	How far data volume moved between the two sides, as a percentage of data points.

Where there is no step, **Before** and **After** report why instead of showing a value, and they distinguish the two reasons carefully: "there was not enough history to look" is a gap, "we looked and there was no step" is a finding, and mistaking one for the other would send you looking in the wrong place.

Below the figures, the **Elapsed vs baseline** chart plots every run oldest first, with a band drawn at 15 percent either side of the median. That band is the same dead band the run list uses, so a point inside it is a run nobody would have flagged.

The note under the chart explains every gap in the line. Runs that recorded no measurement and runs that failed or were canceled are counted separately, because they are different facts. **A break in the line is never a zero.**

Narrow the search to one variation

A job that bursts produces several outputs from one run. Its total moves for reasons that have nothing to do with any one of them: a burst list grew, a run was restricted to a single recipient to test a fix, a region was retired. Every one of those makes whole-run totals incomparable while leaving each surviving output perfectly comparable with its own past.

The **Compare** picker at the top of the tab switches the scope.

1. Leave it on **The whole run** to search the report's total run time.
2. Or select one variation to search only that variation's history. Each entry is one output identity, labeled as it was on the most recent run that produced it.

A variation is what the job author **declared** as varying: a parameter supplied per entry by a burst set, or a parameter set to **one report per item**. It is deliberately not what the data happened to produce, so a weekly report stays the same report every week even when the query behind it returns ten items in January and twenty in June. A

parameter set to **one page per item** shapes worksheets rather than outputs and is not part of the identity, and neither is the recipient, so re-pointing a burst at a different mailbox does not start a new history.

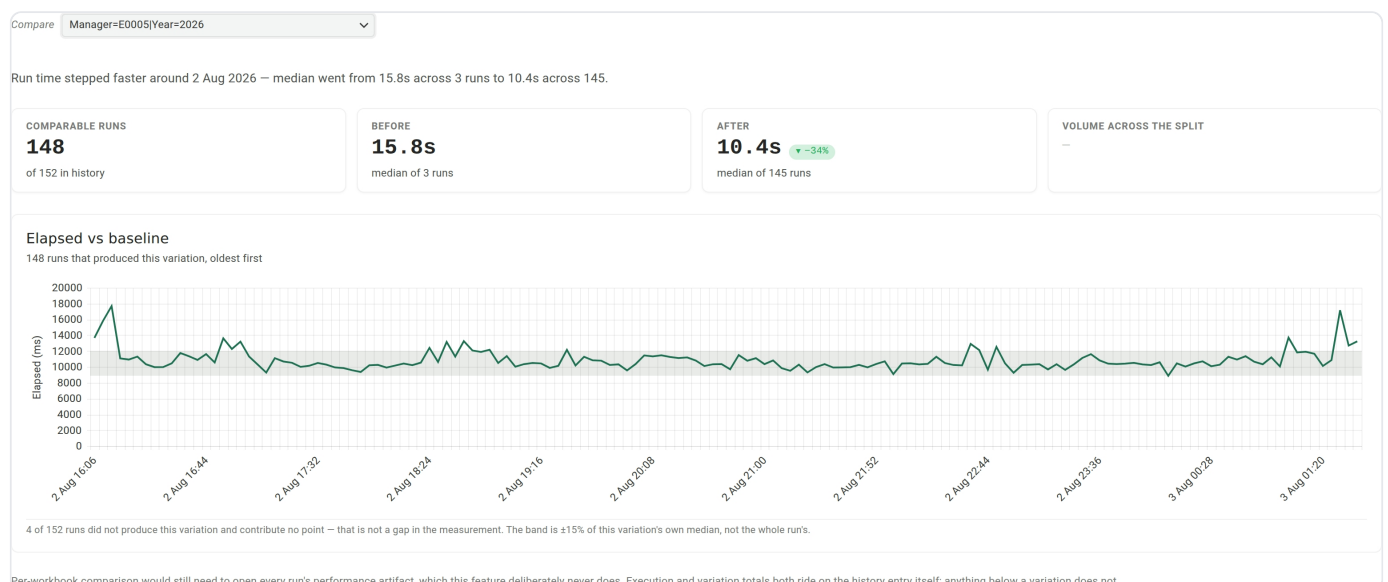
At variation scope, three things change:

- **The chart shows only the runs that produced this variation.** A run that did not produce it contributes no point at all, because the report simply was not asked for that night. The note under the chart says how many runs are absent for that reason, so a twelve-point line under a report with forty runs does not read as missing telemetry.
- **The band is drawn around this variation's own median,** not the whole run's.
- **Release and node are not named.** One variation's history spans executions that may disagree about which node ran them, and naming one would be a coin flip presented as a fact.

Where a single run produced several outputs sharing one identity, their cost is added together into that run's point, and the note says so. Without that, a job whose burst grew from one output to fifty would look like a catastrophic slowdown.

Two notes appear beside the picker when they apply:

- **The job declares no varying parameter,** so every output shares one identity. Trending it is trending the whole job, not one report inside it.
- **The variation has been seen in one run only,** so there is nothing to trend yet.



Notes and limits

- **The picker only appears when there is something to pick.** Variation identity is recorded on runs executed from this release onward, so a report whose runs all

predate it has none. In that case the tab compares whole runs and says so under the chart, rather than offering an empty list that would read as "this report has no variations".

- **A step needs to be worth naming.** The two sides must differ by at least **20 percent**, with at least **3** runs on each side. Below either threshold nothing is reported, deliberately: with less than that, one unusual run wears a confident label.
- **Only one step is reported**, the single most significant one. The tab does not enumerate every wobble in a history.
- **Failed, canceled, and unmeasured runs never take part.** A job that died after 30 milliseconds has a real number and no meaning, and letting it in would either invent a step change or hide one.
- **There is no level below a variation here.** A per-workbook comparison would mean opening every run's performance artifact, which this tab deliberately never does. For a breakdown by output, workbook, phase, or engine, use the **Performance** tab on one run, see [Review a run's performance](#).
- **Comparison ages out with the runs.** History that retention has removed cannot be compared, see [Output retention](#).

Going deeper. To break one run down until the cost is attributable to a phase or a calculation engine, see [Review a run's performance](#). To ask the same question across every report at once, and to pivot by release or by node, see [Compare performance across reports](#).

Scale and size a contribution campaign

[how-to](#)[report author](#)[administrator](#)

A contribution form is not free. Every form a contributor opens becomes a live workbook in a worker process on the server, and that memory is held for as long as the form stays open. A server admits new contributor sessions only while it has memory to spare, so the cost of one form and the size of the server together decide how many people can contribute at the same time.

This guide is about the decisions. For the click-by-click mechanics of each measurement tool, see [Profile and stress-test a campaign](#); this guide explains what to measure, how to read it, what arithmetic turns it into a server size, and how to confirm the answer while a real round is running.

The worked example is a **Q3 budget round**: one budget form per cost center, several hundred contributors, all of whom submit in the last week.

Before you start

- The campaign has generated forms. Every measurement reads the generated form workbooks.
- You can open **Campaign profiling** from the Campaign Dashboard.
- For the live checks in [Watch a live round](#), you need administrator access to **Settings ▶ Performance**.
- For a stress test, prepare a disposable copy of the campaign, a dedicated stress-test user, and one or more Windows load generators, as described in [Profile and stress-test a campaign](#).

The sizing loop

Run these six steps in order. Steps 1 to 3 repeat until the form is as cheap as you can reasonably make it; steps 4 to 6 happen once per deployment decision.

1. **Design the form.** Author it as you want contributors to see it.
2. **Measure its footprint.** Profile one form on the **Memory profiling** tab, and record the numbers.
3. **Tune and re-measure.** Change one thing, profile again, and compare. Only a before-and-after pair on the same machine is meaningful.
4. **Size the deployment.** Combine the measured cost per form with the server's usable memory to get a concurrency number, then a server count.
5. **Prove it under load.** Build a stress plan at the concurrency you sized for, run it, and read what contributors experienced.

6. **Watch the live round.** During the real campaign, confirm on the contribution process console (**Settings ▶ Performance ▶ Contribution**) that the sizing held.

What limits a server

Two separate ceilings decide when a contributor is turned away. They fail differently, so diagnose them separately.

The memory admission valve. When a contributor opens a form, Reportworq checks every online server (the local one and any load-balanced servers) and steers the new session to the online server with the most available memory that is still able to accept one. A server can accept a session only while its memory used is **below 90%**. If no server qualifies, the form does not open and the contributor sees:

Unable to launch new Contribution session because all servers are at capacity.
Please try again later

This is a hard refusal, not a warning or a queue. It is the reason per-form cost matters: usable capacity is bounded by that 90% wall, not by the machine's total RAM.

The per-worker session cap. Contribution sessions run in worker processes, and **Max Sessions per Worker** (on **Settings ▶ Configuration ▶ Performance**, under **Contribution**) limits how many sessions one worker process hosts. Its default is **1**, so by default every live form is its own process. If workers report **At capacity** while server memory is still comfortable, the session cap is binding, not the form's footprint. See [Server configuration](#).

What drives a form's cost

Reportworq measures five things about a form's shape. The **Memory profiling** tab reports sheets, cells, and input cells alongside the workbook size; the **Form sizing** table on the Stress testing tab adds the formula count.

Measured property	Where it appears	Why it matters
Cells	Memory profiling, Form sizing	The count the per-cell cost is multiplied by. This is the dominant term.
Sheets	Memory profiling, Form sizing	Cells are counted across every sheet, so an extra sheet adds its whole cell block.
Input cells	Memory profiling, Form sizing	The cells a contributor can edit, and the ones a stress test fills.
Formulas	Form sizing	Formula cells across the workbook. Recalculation is part of what the profiler exercises.
File size	Memory profiling, Form sizing	The generated workbook on disk, a quick relative check between forms.

The profiler reports **Bytes per cell (at peak)**, which is simply the measured peak cost divided by the cell count. Read the relationship in that direction and it stays honest: **peak cost is roughly bytes per cell multiplied by cell count**, so the reliable way to make a form cheaper is to make it smaller. Removing an unused sheet, or trimming a form down to the range contributors actually fill, removes cells and therefore removes cost, and the profiler will show you exactly how much.

Two things the measurements do *not* tell you, so treat claims about them as judgment rather than fact:

- The profiler measures the whole form's footprint. It does **not** attribute cost separately to formulas, formatting, charts, or images, so it cannot confirm that removing any one of those saves a given amount. If you think one of them is expensive, remove it, profile again, and let the before-and-after numbers decide.
- Bytes per cell varies between forms. A form whose bytes-per-cell figure is far above your other forms is worth investigating, but the number does not say what is responsible.

One campaign setting is a genuine, verifiable memory lever. **Manual calculation mode** (in the campaign's **Options** section) stops the form recalculating on every edit; contributors press **Recalculate** instead. Under manual calculation the server releases the session's workbook 30 seconds after the last action that needed it, and rebuilds it on demand when the next one arrives, so an idle contributor who has left a form open costs far less than one in automatic mode. The trade is that a contributor waits for the reload on their next action. For large forms filled slowly over a day, that trade is usually worth making. See [Author a contribution campaign](#).

Measure the footprint

Use the **Memory profiling** tab in this order. The click path for each run is in [Profile and stress-test a campaign](#).

1. **Profile your heaviest single form first.** Select one form, leave **Flip through sheets** and **Enter sample data** on, and run it. This gives the detailed per-form footprint.
2. **Tune, then re-run the same profile.** Change one thing at a time and compare against the first run. Do the comparison on the same machine; the absolute numbers move with hardware, the deltas do not.
3. **Profile several forms together.** Select the forms a real round will have open at once and run again. This is the run that produces the marginal cost per form, and it is the number the sizing arithmetic uses.

Read the results like this:

Number	What it tells you	How to use it
Peak per form (avg)	The working memory one open form costs at its busiest, averaged over the scored cycles.	The headline cost of a form. Compare before and after every change.
Steady (just opened)	The cost right after the form opens, before any interaction.	The floor a form costs simply by being open, which is what most contributors hold most of the time.
Bytes per cell (at peak)	Peak cost divided by the cell count.	A size-independent comparison between forms. Use it to spot a form that is expensive for its size.
Held after close	Memory still held after the form closed, measured against the idle floor.	A retention signal. A small figure is normal; a figure that climbs cycle after cycle is not.
Worker baseline	What a worker process costs before any form is open.	Add it per session when workers host one session each, which is the default.
Marginal cost per form	The true cost of each additional concurrent form, measured after the first.	The concurrency number. Use this, not the first-form cost, for the arithmetic.
First form	The first concurrent form's cost, which carries one-time warm-up.	Do not multiply this. It is a one-off.

The single-form result also shows an estimate reading **~N of this form open at once on the profiling machine**. It is derived from the free memory on the profiling machine,

taking 80% of it and dividing by the average peak per form. Treat it as **indicative only**: it reflects the machine that ran the profile rather than production, it ignores the worker baseline, and it ignores the per-worker session cap. Use it as a sanity check on the arithmetic below, not as the answer.

Turn the measurements into a sizing answer

Work through the arithmetic with your own measured numbers. The figures below are illustrative.

What was measured for the Q3 budget round:

- Peak per form (avg): **180 MB**
- Worker baseline: **100 MB**
- Marginal cost per form: **120 MB**

What the server offers:

- Total memory: **64 GB**
- Memory in use when no forms are open (operating system, Reportworq, everything else): **12 GB**

Step 1: decide the memory you will plan to. The valve refuses at 90% used, so never plan to consume the full 90%. Plan to **80%** and keep the last ten points as headroom for spikes and for other work on the machine.

80% of 64 GB = 51.2 GB
minus the 12 GB idle floor = 39.2 GB (40,140 MB available for sessions)

Step 2: work out the cost of one concurrent contributor. This depends on **Max Sessions per Worker**.

- At the default of **1**, each session is its own worker process, so it pays the baseline itself: 100 MB baseline + 180 MB peak per form = 280 MB per contributor.
- With sessions multiplexed at, for example, **10 per worker**, ten sessions share one baseline: $(100 \text{ MB} + 10 \times 120 \text{ MB}) / 10 = 130 \text{ MB per contributor}$.

Step 3: divide.

Default (one session per worker): $40,140 / 280 = 143$ concurrent forms
Multiplexed (10 per worker): $40,140 / 130 = 308$ concurrent forms

Step 4: convert concurrency into contributors. Nothing measures how many of your contributors are online simultaneously; you have to estimate it from how the round

behaves. If roughly one in four of the roster is filling a form at the busiest moment, a server carrying 143 concurrent forms supports a roster of about 570 people.

Step 5: convert contributors into servers. For a roster of 1,200 at the same one-in-four peak, the round needs about 300 concurrent forms. At 143 per server that is 2.1 servers, so provision **three**. Always round up: the consequence of being one server short is contributors being refused, not contributors waiting.

The assumptions this arithmetic rests on. State them alongside the answer, because each one can move it:

- The measurements come from the machine that ran the profile. Re-measure on hardware that matches production before committing to a server count.
- The idle floor is whatever the server is already using. Read it from the live per-server percentage on the contribution process console rather than assuming it.
- The peak-concurrency ratio is your estimate, not a measurement.
- Contribution is not the only load. Report and job execution take memory from the same machine at the same time.
- Planning to 80% rather than 90% is deliberate. The valve is a refusal, so the margin is what stops a normal spike becoming an outage.

Prove the experience with a stress test

The arithmetic says how many forms fit. A stress test says whether contributors have an acceptable experience at that number. Build the plan on the **Stress testing** tab so the run answers the sizing question:

- **Concurrent forms / node** multiplied by **Nodes (machines)** should add up to the concurrency you sized for. The **Intended peak concurrent** stat on the plan preview shows what the plan will actually attempt.
- **Nodes** matter because the forms are partitioned across them; each form runs on exactly one node. If you need more concurrency than one load generator can drive, add nodes rather than raising the per-node number indefinitely.
- **Editor pace** decides whether the run is a burst or a soak. Slide toward **Slow & deliberate** for a believable round where contributors hold forms open for a long time, which is the shape that actually tests memory. Slide toward **Fast** only when you are testing throughput rather than capacity.
- Leave the **Advanced settings** alone unless you have a reason. The defaults fill 50% to 75% of each form's input cells and submit 25% of the time, which is closer to a real round than anything you are likely to invent.

Then read the **Analysis** tab against these questions:

- **Did the run reach the concurrency you sized for?** Check **Peak concurrent forms** in the Run summary. If it is well below the intended peak, the run did not test what you meant to test.
- **Were there errors?** The **Actions** stat carries the error count. Errors at load are a capacity answer, not a detail.
- **What did contributors feel?** The experience table gives each action in seconds as **Typical** (half of users), **Most users** (9 in 10), and **Slowest seen** (the worst single measurement). Opening a form is the number contributors judge the system by; entering a value, recalculating, and changing worksheets are the ones they feel repeatedly.
- **What did Reportworq flag?** A **slow tail** finding means the slowest 1 in 100 measurements took at least three times the typical time, so most contributors were fine and a few were not. A **slowdown** finding means an action in the last third of the run averaged at least 1.5 times the first third, so something accumulated as the run went on.

Read the outcome this way:

- **This scales** when the run reached the intended peak, errors are negligible, in-form actions stay around a second or less, form opens stay acceptable for your users, and there are no findings.
- **This does not scale** when the run produced errors, when a slowdown finding appears (the deployment degrades the longer the round runs, so a real multi-day round will be worse), or when the slow tail is severe enough that a meaningful share of contributors would be waiting.

If a run fails, halve the concurrency and run again. The concurrency at which the findings disappear is your real per-server number, and it replaces the arithmetic.

Watch a live round

Settings ► Performance ► Contribution is the operations console for the contribution runtime. It shows the present moment only, refreshing every 10 seconds, and it keeps no history. There are no timings, no percentiles, and no record of yesterday. It also identifies a session by input-form name only, so with two campaigns live at once you cannot attribute load to one or the other from this screen. Use it to confirm that the sizing held and to act when it did not, and use Campaign profiling for anything you need to measure or keep.

Full field-by-field detail is in [Contribution processes](#). During a live round, three readings matter:

- **The per-server memory percentage** in the left rail, next to each online server. This is the number the admission valve reads. Watching it approach 90% is the warning that contributors are about to be refused.
- **Sessions-alive** in the selected worker's memory breakdown. It is the worker's working set minus its idle baseline, so it is the cost of the live sessions on that worker. Divide it by the worker's active session count and you have an empirical megabytes-per-contributor figure to check your sizing against. Two caveats: the baseline reads 0 until the worker has reported it, and the figure covers every session on the worker, so it is only a clean per-form number when one form is in play.
- **The worker state. At capacity** means the worker has reached **Max Sessions per Worker**, which is a session-count limit and has nothing to do with memory. The other states are Active, Draining, Warm, Idle, Legacy 1:1, and Stopped.

Two readings on this screen are easy to misread:

- **Headroom is not free machine memory.** It is the distance between this worker and the busiest worker on the same server. A large headroom figure does not mean the server has room.
- **The memory bars are scaled per server**, relative to that server's busiest worker. Bars on different servers are not comparable to each other.

When the valve starts refusing sessions, free memory now and re-size afterwards:

1. Select the server whose memory percentage is highest, then its hottest worker.
2. **Kill session** on a session that is clearly abandoned. The contributor's unsaved edits are recorded and the form lock is released, so this reclaims a stuck session without losing work.
3. **Force GC** on the worker to reclaim what it can.
4. **Drain** a worker so it stops taking new sessions and exits when its current ones end, or **Shutdown** it if you accept ending its live sessions.
5. After the round, treat the refusal as a sizing failure, not an incident. Re-measure the form, redo the arithmetic, and add memory or a server.

Red flags and what to do

Symptom	Likely cause	What to do
Contributors are told all servers are at capacity	The deployment is undersized for the peak, or the round peaked higher than estimated	Free memory on the hottest server now, then redo the sizing arithmetic with the real peak and add memory or a server
Workers read At capacity while server memory is comfortable	The per-worker session cap is binding, not memory	Raise Max Sessions per Worker on Settings ► Configuration ► Performance, then re-measure the per-session cost, which falls when sessions share a worker baseline
Held after close climbs cycle after cycle	The form is retaining memory after it closes	Re-run the profile to confirm, download the CSV, and raise it with support. Until it is resolved, size on the retained figure as well as the peak
Bytes per cell far above your other forms	The form is expensive for its size, for a reason the profiler does not attribute	Remove one suspected element at a time and re-profile. Let the deltas identify the cause
A slow tail finding on opening a form	Contention at peak concurrency, so most contributors are fine and the unlucky ones wait	Re-run at half the concurrency. The level at which the finding disappears is the per-server number to size to
A slowdown finding on any action	Something accumulates over the run, so a long round degrades	Do not open the round at this size. Re-run at lower concurrency and confirm the finding is gone
High Served (lifetime) against a short Uptime	Workers are recycling often, so contributors repeatedly pay a cold start	Review Recycle Worker After Sessions on Settings ► Configuration ► Performance; its default is 25
A server shows a last seen note instead of workers	The server has not reported for more than 5 minutes and is treated as offline	Check the server. While it is offline it takes no sessions, so the remaining servers absorb the whole round

Notes and limits

- Memory profiling is read-only and never changes campaign state. Stress testing writes data and submits forms, so it must run against a disposable campaign that is reset before every run.

- Profiler numbers are working set on the machine that ran the profile. Deltas between runs are dependable; absolute numbers are only as representative as that machine.
- The contribution process console is live only. Capture anything you want to keep at the time you see it.
- The concurrency estimate on the Memory profiling tab ignores the worker baseline and the per-worker session cap, so the arithmetic in this guide will usually give a lower and more realistic figure.

Related topics

- [Profile and stress-test a campaign](#)
- [Author a contribution campaign](#)
- [Contribution \(write-back\) overview](#)
- [Contribution processes](#)
- [Deployment topology](#)
- [Performance and capacity planning](#), for how contribution sizing sits beside report and job execution on the same servers

Job options: collation and splitting

AUTH-03

how-to

report author

Job Options is a control in the Job Editor's chrome band, alongside the report title, Save, and Discard. It sets **report-level** output behavior: how the job's worksheets are assembled into files. These are job-wide settings, distinct from the per-worksheet settings on the Reports step (see [Worksheet output options](#)).

Before you begin

- Have a report with at least one worksheet. Job Options is available throughout authoring from the editor chrome.

Set the options

1. In the Job Editor chrome, select **Job Options**.
2. Set either option:

Option	What it does
Enable manual collation	Turns on manual control over how worksheets and outputs are collated (grouped and ordered) into the produced files, instead of the default automatic collation.
Split worksheets	Splits the job's worksheets into separate outputs, for example one file per worksheet, rather than combining them into a single workbook.

Split worksheets is how you produce CSV. A CSV file holds exactly one sheet of data, so a multi-worksheet job has to be split before it can be written out as CSV. Turn on **Split worksheets** and select the CSV output format, and each worksheet becomes its own `.csv`. Leave it off and only one worksheet can survive the conversion. See [Output formats](#).

These options are also surfaced as **Manual Collation** and **Split Worksheets** in the Distribution step's advanced options.

When to use each

- **Enable manual collation** when the order in which worksheets are packaged into the delivered file matters and the default assembly is not right, for example forcing statement, then supporting schedules, then notes in a monthly pack. Manual

collation also works with "one page per item" replication so pages can interleave by parameter across worksheets (see the replication modes in the [Parameters reference](#)).

- **Split worksheets** when each worksheet should be delivered as its own file rather than as one combined workbook, for example one file per department, or one file per cost-center worksheet for individual distribution.

Notes and limits

- Job Options are job-wide. Per-tab naming, color, ordering, and suppression are handled by the per-worksheet [Worksheet output options](#).
- Format-specific packaging such as zipping outputs and combining emails is set on the Distribution step, not here.

Related topics

- [Worksheet output options](#)
- [Parameters reference](#)
- [Choose the source workbook and worksheets](#)

Combine emails, collate, and zip

DIST-02

how-to

report author

These options shape how the finished files are packaged **before** delivery: whether a recipient gets one combined email or several, whether pages are sorted, and whether outputs arrive as a single zip. They are underused, mostly because it is not obvious *when* you would reach for them, so each section below leads with the use case.

What they do. Three levers work together on the Email distributor:

- **Combine email messages when possible** merges outputs that share the same addressing into one message.
- The **Combine Key**, an optional extra field, gives you one more level of control over that grouping when matching addresses alone do not split the mail the way you want. It is covered in detail below.
- **Zip Files** bundles the attachments of a message into a single archive.

Two further options sit in the **Job Options** sidebar and change the files themselves rather than the mail: **Enable Manual Collation** sorts report pages, and **Split Worksheets** turns one workbook into one file per tab.

Before you begin

- You are on the **Distribution** step, with output formats selected and at least the **Email** distributor added (the combine and zip options are Email-distributor options).
- Collation and Split Worksheets live in the **Job Options** sidebar; the combine and zip options live on the Email distributor's options sidebar, and the Combine Key sits on the Email distributor itself.

Combine emails: one message instead of many

By default, a job that produces several outputs (or bursts to several recipients) can send a separate email per output. **Combine email messages when possible** merges outputs that belong together into a **single email**.

What it does. It defers the individual emails until the batch of outputs has been generated, groups the messages that share the same **To, Cc, Bcc, From, and Combine Key**, and then sends one message per group carrying all of that group's attachments. Address matching ignores case and ordering, and the subject and body do not have to match.

When to use it. A recipient should get **one** email with their whole pack, not five. For example, a controller who is due the P&L, the balance sheet, and the cash-flow report should receive a single message with three attachments, rather than three separate emails.

To turn it on, select **Combine email messages when possible** on the Email distributor's options.

The combine key: control what gets grouped together

Grouping is by addressing alone unless you say otherwise. The **Combine Key** is a free-text, tokenizable field on the Email distributor that is folded into that grouping, so two messages combine only when their addresses **and** their combine keys match. It appears once combining is on; select **Show More Fields** on the Email distributor to reveal it. Because it accepts variable tokens, its value can resolve per burst iteration.

When to use the combine key. Reach for it when you need one more step of control over how messages are split than the recipient's address gives you.

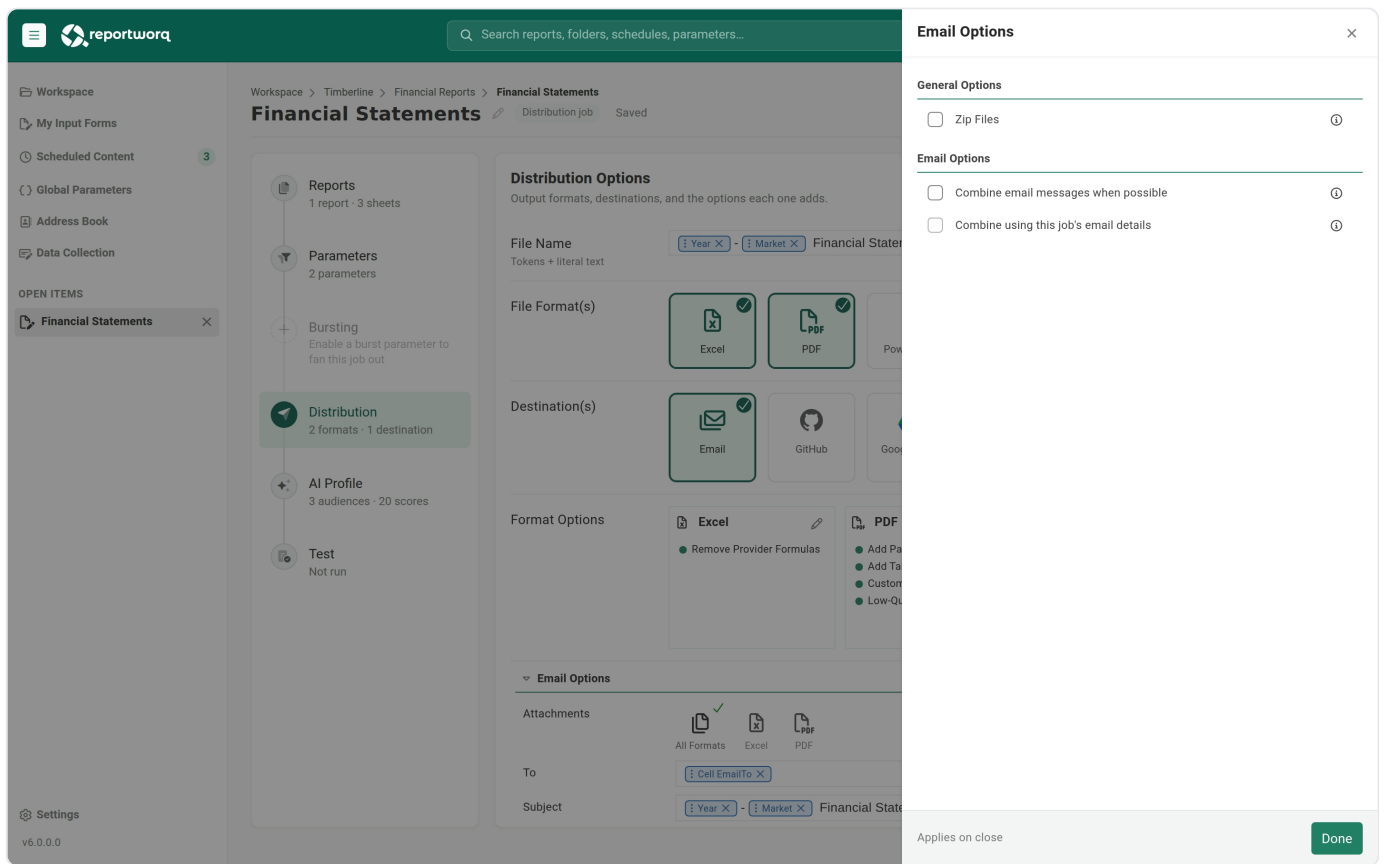
Take an income statement bursted to one PDF per store, addressed to the store's district manager. District Manager A covers three districts of 6 to 12 stores each. Her email address is the same on every one of those stores, so combining on addressing alone gives her a single message with up to 48 attachments, all mixed together. Put the **District** parameter into the **Combine Key** and the run groups by manager *and* district instead: she receives three emails, one per district, each carrying only that district's store files. The files arrive already separated the way she works with them.

The same lever also works in the other direction. A shared combine key can pull outputs from **more than one job** in the same scheduled run into one message, for example a P&L job and a variance-commentary job that should reach each region's manager together.

The trap: only one job supplies the combined email body

When you combine across multiple jobs, exactly **one** job may be designated to supply the combined email's content and options, using the **Combine using this job's email details** setting. Set it on that **single** job only.

Two jobs cannot both "combine into this email body". If more than one job in the run claims to provide the combined email's details, the combination is ambiguous. Pick the one job whose subject and body should win, and leave the setting off on the others.



Collation: sort report pages by cell value

Enable Manual Collation sorts the report's pages based on cell values before output.

When to use it. The pages of a compiled pack need to come out in a specific order driven by data (for example a ranking, or a defined regional sequence) rather than worksheet order.

Zip Files: bundle outputs into one archive

Zip Files (an Email distributor option) bundles the run's outputs into a single `.zip` attachment.

When to use it. A recipient should receive one archive instead of a handful of loose attachments, or the total attachment size is easier to send zipped.

The **Exclusion List** is a semicolon-separated list of file names or extensions to leave **out** of the zip (for example `.pdf; data.xlsx`). It only has any effect when **Zip Files** is on.

Split Worksheets: one file per tab

Where combining and zipping *group* outputs, **Split Worksheets** (in Job Options) *fans them out*: it saves each **visible** worksheet as a separate output file. Every selected output format is then produced per worksheet, so a three-tab report with CSV selected yields three `.csv` files rather than one. Each is named from the output file name plus the worksheet name, for example `Sales Extract East.csv`.

When to use it. A downstream system needs **one flat file per table**. A CSV holds a single sheet of values, so without Split Worksheets a multi-tab workbook exports only its first worksheet and the rest of the data never reaches the target. Turn Split Worksheets on and every tab arrives as its own `.csv` for the load job to pick up: actuals, budget, and headcount detail each become a separate file. The same applies to any integration that expects one file per table rather than one workbook.

Split Worksheets fans out **worksheets**; **bursting** fans out separate runs per parameter value, a different mechanism (see [What bursting is and when you need it](#)).

Use cases

- **Single zipped email to a controller.** Zip Files on, Exclusion List drops the raw `.xlsx`, combining on, so one message carries the full pack.
- **One email per district for a multi-district manager.** Combining on with the **District** parameter in the Combine Key, so each district's store PDFs arrive in their own message.
- **One email per regional manager across two jobs.** Combining plus a shared combine key so a P&L job and a commentary job reach each manager in one message; only the P&L job carries the email details.
- **Per-table CSV extract.** Split Worksheets on with CSV selected, so each tab of the workbook ships as its own `.csv` for a downstream load.

Notes and limits

- **Combining defers the send.** Individual outputs wait until the batch completes, then go out combined. A run that is waiting on the batch is not finished until the combined mail is sent.
- **Grouping is by To, Cc, Bcc, From, and Combine Key.** Two messages combine only when all five match. Subject and body are not part of the match. The addresses are matched without regard to case or order, but the Combine Key is matched exactly, so keep its casing consistent if you type it by hand rather than build it from a token.
- **Exclusion List needs Zip on.** It does nothing unless Zip Files is enabled.

- **One combine-details job.** Across a multi-job combined run, only one job may supply the combined email's subject, body, and options.
- **Split Worksheets covers visible worksheets only.** A hidden tab produces no file, so a calculation or lookup tab stays out of the fan-out.

Going deeper. Write the combined message's subject and body once, with dynamic tokens, and reuse it across runs. See [Email messages and templates](#).

Anaplan functions

reference

report author

The Anaplan connector contributes a single worksheet function, `ANAPLANINPUT`, which marks a cell as a write-back input so its value is posted to the matching Anaplan module cell during contribution processing.

Unlike the rest of the function surface, `ANAPLANINPUT` **has no dotted canonical alias**. There is no `RW.Anaplan.*` name in the current build; `ANAPLANINPUT` is the function's only registered name.

ANAPLANINPUT

Signature

```
=ANAPLANINPUT(inputCell, [tuplePart1], [tuplePart2], ...)
```

#	Argument	Type	Meaning
1	<code>inputCell</code>	Cell reference	The cell whose value is written back to Anaplan.
2 and up	<code>tuplePart</code>	String	Optional. Extra arguments that flag the formula as carrying its own tuple. See the warning below before using them.

How the write-back actually finds its target

This is the part worth understanding, because it explains every failure mode.

1. The formula marks the cell as an **input**, so it is processed during **input submission**, not during the report-output pass.
2. **During report generation**, Reportworq resolves the Anaplan tuple behind the input cell, which is the workspace, model, module, dimensions and members that make up that intersection, and stores it as a worksheet custom property named `ANAPLANINPUT_{row}_{col}`. This is the "breadcrumb".
3. **At submission**, the breadcrumb is what Reportworq posts against. Inputs are grouped by their `workspace | model | module` key, and each group is posted as a single mutation request.
4. On success the formula cell is cleared.

The breadcrumb is only written when the input cell sits on a cell that an Anaplan view generated. If the cell does not point at an Anaplan cell, report

generation logs an error for that input, and there will be nothing to post later.

The trailing tuple arguments are a trap

The optional `tuplePart` arguments do not do what their name suggests.

They are recorded, and their presence **suppresses** automatic breadcrumb generation for that cell. But in the current implementation the write-back vector is built **only** from the breadcrumb. The arguments are never used to construct it.

The result is that supplying them turns off the only mechanism that would have found the target, and the input then has no tuple to post against. **Omit them** unless you know a breadcrumb already exists for the cell.

This differs from the Workday Adaptive equivalent, `RW.Adaptive.Input`, where explicit tuple parts *are* used to build the vector.

Examples

```
=ANAPLANINPUT (B5)
=ANAPLANINPUT (D12)
```

The simple form is the correct form.

When an argument is wrong

Situation	Result
<code>inputCell</code> is not a real cell reference	Formula error during parsing.
The input cell is not on a cell an Anaplan view generated	No tuple breadcrumb is written, and report generation logs an input error.
Trailing <code>tuplePart</code> arguments supplied, expecting them to define the target	Breadcrumb generation is skipped and no vector is built from the arguments, so the write-back has no tuple to post.
The breadcrumb is missing at submission time	Submission fails for that input with a missing-tuple error.

Placement

Place `ANAPLANINPUT` on a cell that a refreshed Anaplan view produced, so Reportworq can capture the tuple behind it. The formula relies on the workbook being a recognized Anaplan report.

Keep related inputs in the same report run: multiple input cells targeting the same workspace, model and module are batched into one post.

Keep write-back formulas in hidden rows or columns. Like the other write-back families, they render as `#NAME` in a contributor's form, so their rows and columns are hidden by design.

Notes and limits

- The function is inert in desktop Excel.
- It ships with the Anaplan connector plugin. A specific introducing version is not determinable from the current source.

Related pages

- [Functions overview: which one do I use](#)
- [Contribution functions](#) for the input form the write-back sits behind.
- [Workday Adaptive functions](#) for the equivalent tuple-based write-back, which behaves differently.
- [Connector catalog](#) for the Anaplan connection setup.

Save and Save As

AUTH-03

how-to

report author

The Job Editor chrome (the band across the top of the editor, alongside the report title, **Job Options**, and **Discard**) carries the **Save Changes** control. Selecting the caret beside it opens **Save As**, which saves a copy of the report under a new name instead of overwriting the one you have open.

Before you begin

- Have a report open in the Job Editor with unsaved or previously saved content. Save As works whether or not you currently have unsaved changes.
- Know the folder the report lives in - the new copy is created **alongside the original**, in the same folder.

Save Changes (unchanged)

Selecting **Save Changes** writes your edits back to the report you have open. The report keeps its name and identity; only its content is updated.

Save As: create a copy

1. In the Job Editor chrome, select the caret beside **Save Changes**.
2. Select **Save As....**
3. Enter a name for the copy. The editor suggests `Copy of <current name>`; replace it with anything that is not already used by another report in the same folder.
4. Confirm. Reportworq creates a new report with that name from the report's **current** content - including any changes you have made in this editing session, saved or not - and the Job Editor switches to editing the new copy.

The original report is left exactly as it was last saved. Save As never overwrites it and never renames it. After Save As completes, **Save Changes** in this editor tab now saves to the new copy, not the original - if you want to keep editing the original report, open it again from the workspace.

When to use it

- **Create a variation** of an existing report (a different region, department, or format) without touching the one already in production.
- **Checkpoint before an experiment.** Save As before making a risky change, so the last-known-good version stays available under its own name if you want to go back to it.

- **Start a new report from an existing one** that already has most of the source, parameters, or distribution setup you need, rather than building from scratch.

Notes and limits

- The new name must be unique within the destination folder; Reportworq rejects a name already used by another item there.
- Save As always places the copy in the **same folder** as the original. Move or rename it afterward from the workspace if it belongs somewhere else.
- Run history, schedules, and audit log entries belong to the original report and are not copied - the new report starts with a clean history.

Related topics

- [Job options: collation and splitting](#)
- [Sign in and take the tour](#)

The Workspace Inbox

[how-to](#)[report author](#)[administrator](#)[user](#)

The **Workspace Inbox** is a mailbox built into Reportworq. Turn it on and Reportworq keeps a copy of **every message it sends**, viewable in the app through a dedicated **Email Inbox** screen that looks and behaves like a lightweight web mail client.

It runs **alongside** your real email provider. It does not replace one. That is what makes it useful in day-to-day operation, not just in a demo:

- **Authors** can see their own distributions, exactly what went out, with the real subject, body, and attachments, without needing access to a recipient's mailbox.
- **Administrators** can review delivery history from a **mailbox perspective**: open any recipient's inbox and see what that person received, and when.

It is also the way to run Reportworq **with no mail server at all**. Enable the Workspace Inbox, leave the email provider set to **None**, and jobs run their full distribution path with every message landing in the inbox instead of the internet. That suits trials, demos, and training.

Previously called the "Demo" email provider. In earlier releases this was a *provider* you selected instead of a real one, so you could either send mail or inspect it, never both. It is now a checkbox that works with any provider. If you upgrade an install that had the Demo provider selected, the Workspace Inbox is switched on for you automatically and nothing is lost.

Turning it on

1. Go to **Settings > Configuration > Email**.
2. Choose your **Authentication Provider** as normal: Basic, Microsoft 365 SMTP, Microsoft 365 Graph, or SendGrid. Or leave it on **None** if you do not want anything sent.
3. Under **Workspace Inbox**, select **Enable Workspace Inbox**.
4. Save the configuration.

An **Email Inbox** button (an inbox icon) appears in the top-right app bar. Selecting it opens the Email Inbox screen.

- The button shows for **every signed-in user**, readers, authors, and administrators alike, so everyone can open their own captured mail.

- It appears the **next time the page loads** after saving, so refresh your browser if you do not see it immediately.
- It is hidden again if the Workspace Inbox is switched off.

You do not need a real email provider. With the inbox enabled, Reportworq considers email configured, so distribution steps, notifications, and password-reset mail all work. They just land in the inbox instead of being transmitted.

How messages get captured

You do not capture messages manually. Running work does it for you.

1. With the Workspace Inbox enabled, run (or let a schedule run) any job or step that sends email, for example a distribution step that emails a report, or a burst that emails a report to many recipients.
2. Reportworq builds the full email, stores a copy in the Workspace Inbox, and then hands it to your email provider for delivery, if one is configured.
3. The message appears on the Email Inbox screen, newest first.

One copy per recipient. A captured message is stored once for each distinct recipient across the **To**, **Cc**, and **Bcc** lines. A report burst that emails three people produces **three independent inbox entries**, one in each person's inbox, just like real mail. Read and unread state and deletion apply per inbox, so marking a message read in one recipient's inbox does not affect the others.

Each entry keeps a complete copy of the original message (an `.eml` file) including all attachments, so you can open it, read the body, and download everything later.

Using the Email Inbox screen

The screen is a two-pane, master and detail layout:

- **Left, the message list.** One row per captured message, newest first. Line 1 shows the sender and a relative time, line 2 shows the subject, with a **paperclip** icon when the message has attachments. Unread rows are **bold** with a leading accent dot. A message count, and unread count, sits in the footer beneath the list.
- **Right, the reading pane.** Shows the selected message's subject, the From, To and Cc headers and full timestamp, the message body, and its attachments.

At the top right of the screen are **Refresh**, **Clean inbox**, and, for administrators, **Clear all inboxes**.

Read a message

Select any row in the list. The message opens in the reading pane and is **marked read automatically**. The body renders as formatted HTML, with inline images shown in place, or as plain text, matching how the email was composed.

Download attachments

In the reading pane, each attachment appears as a **chip** showing its file name, size, and a file-type icon.

- **Download one attachment:** select its chip. The whole chip is the download button.
- **Download every attachment:** open the reading pane's **Actions** menu and choose **Download all attachments**. Reportworq packages them into a single `.zip`.

Download the whole message

To keep or forward the raw email, open the **Actions** menu and choose **Download .eml**. This saves the complete message, headers, body, and attachments, as a standard `.eml` file you can open in Outlook, Apple Mail, or any mail client.

Mark read or unread

Two ways:

- **From the list:** every row has an always-visible envelope toggle. Select it to flip a message between read and unread without opening it.
- **From the reading pane:** open the **Actions** menu and choose **Mark read** or **Mark unread**.

Delete a message

- **From the list:** select the trash icon on the row.
- **From the reading pane:** open the **Actions** menu and choose **Delete**.

Deletion asks you to confirm and **cannot be undone**. The message and its attachments are removed permanently.

Clean the inbox

Clean inbox permanently removes **all** captured messages from the inbox you are currently viewing, attachments included. You are asked to confirm, and the action cannot be undone. The button is disabled when the inbox is already empty.

Refresh

Refresh reloads the inbox list, which is useful after running a job in another tab so newly captured messages appear.

What you can do by role

The Email Inbox is scoped to who you are.

Reader or author: your own inbox

A reader or author sees **only their own inbox**, the messages captured for the email address they signed in with. There is no inbox picker, the list header simply shows your address. You can read messages, download attachments and `.eml` files, mark read and unread, delete individual messages, and **Clean inbox** to empty your own mailbox.

System administrator: every inbox

A **system administrator** gets two extra capabilities:

- **Inbox picker.** A menu at the top of the message list lists **every recipient address** that has captured mail. Pick any address to view that user's inbox. This is how an administrator answers "what did this person actually receive, and when", in production as well as during a demo or training session.
- **Clear all inboxes.** A destructive action that empties **every** inbox for **every** user in the tenant at once. Because it is irreversible and tenant-wide, it requires an explicit confirmation: you must **type CLEAR** to proceed. This is separate from **Clean inbox**, which only empties the one inbox you are viewing.

Where it captures from

The Workspace Inbox captures the same mail your email provider sends, so it works with the ordinary distribution flow:

- Email **distribution steps** on a job.
- **Bursts** that email a report to a list of recipients, where each recipient gets their own captured copy.
- Any **scheduled** run of the above.

The subject and body come from the same **message template** the real transport would use, so what you see in the inbox matches what a recipient would have received.

Limits

A captured copy is not a delivery receipt. Reportworq stores the copy *before* handing the message to your email provider, so a message can appear in the inbox and still have failed to send, for example if the mail server was unreachable. Check the **job log** for the delivery outcome. The inbox tells you what was *sent*, not what was *received*.

- **Capture and view only.** The Email Inbox has no compose, reply, forward, or folders. It only receives and displays the mail your jobs produce.
- **Manual cleanup only.** Captured messages persist until you clear them, via **Clean inbox** or an administrator's **Clear all inboxes**. There is no automatic expiry or retention window, so on a busy installation, plan to clear the inbox periodically.
- **Per-recipient copies.** A message to several recipients creates several independent entries. Deleting or cleaning one recipient's inbox does not touch the others.
- **Optional secondary .eml folder.** If an administrator has set **Also Save Copies To** in the email configuration, Reportworq drops a copy of each message into that folder as well. The Email Inbox works with or without it.

Troubleshooting

Symptom	Likely cause	What to do
No Email Inbox button in the app bar	The Workspace Inbox is not enabled, or the page has not reloaded since it was	In Settings > Configuration > Email , select Enable Workspace Inbox and save, then refresh your browser.
Ran a job but the inbox is empty	The job step did not send email, or the inbox was not enabled when it ran	Confirm the step is an email distribution, or a burst that emails, and that the Workspace Inbox was enabled during the run. Select Refresh .
Recipients are not receiving real emails, though the inbox has copies	The Authentication Provider is set to None	Set a real provider for actual delivery. The Workspace Inbox keeps working alongside it.
A message is in the inbox but the recipient never got it	The copy is stored before the send is attempted, so the send may have failed	Check the job log for the delivery error, then verify your email provider settings with Send test email .
A reader cannot see another user's messages	Per-role scoping: non-administrators see only their own inbox	This is by design. Only a system administrator can view other users' inboxes, via the inbox picker.
A burst to several people shows separate copies	Per-recipient capture, by design	Each recipient has an independent inbox entry. An administrator can switch between them with the inbox picker.

Related

- [Destinations](#) for the Email destination whose sends this captures.
- [Email messages and templates](#) for the subject and body of captured mail.
- [What bursting is and when you need it](#) for per-recipient distribution.

PowerPoint reporting

[how-to](#)[report author](#)

Reportworq turns a PowerPoint template into a delivered deck on every run, and, if you want, a PDF companion alongside it. The slide design stays yours. Reportworq refreshes the data-bound content each time the job runs.

The template is not a report source. Your Excel workbooks are the sources, on the Reports step, because they hold the data. The `.pptx` is chosen on the **Distribution** step, as part of the PowerPoint output format, because it describes how that data is presented.

A template on its own is only a shell. It becomes a data-driven deck when each slide shape is **linked to an object in the underlying Excel report**: a cell range, a named range, a chart, or a text box. At run time Reportworq replaces each linked shape's contents with fresh data.

When to use it. Reach for PowerPoint output when your audience needs a formatted presentation, a board deck, an executive review, or a regional pack, rather than a spreadsheet or a PDF report. For tabular or pivot-heavy detail, an Excel or PDF report is the better fit.

Before you start

- **Build the job in Reportworq first.** A shape link points at a report object *inside a specific job*, so the job must already exist and be saved before a template can be linked to it. Work in the order below: sources, then output format and template, then save, then link the slides.
- A `.pptx` template reachable through a report provider, typically a network-folder provider. PowerPoint output cannot be generated without a template.
- The Excel report or reports whose data the slides will show. These are added as sources on the Reports step.
- The report objects the slides will show, meaning the charts, named ranges, text boxes, and cell ranges, must already exist in those Excel reports. The add-in lists what is there; it does not create objects for you.
- The Reportworq add-in installed in PowerPoint, **HTTPS on the Reportworq server**, and a signed-in licensed account. See [The Office add-ins](#).
- Sign-in **in the same workspace as the job**. A job in another workspace does not appear in the add-in's job picker.

Add the Excel sources

On the **Reports** step, add the Excel report or reports whose data the slides will show. These are the job's sources. See [Choose the source workbook and worksheets](#).

Do not add the `.pptx` here. You select it in the next step.

Select PowerPoint as an output format and pick the template

1. Go to the **Distribution** step.
2. Select the **PowerPoint** output format. Its options sidebar opens.
3. Set the **Template File** with **Select a PowerPoint File**. This browses your report providers, filtered to PowerPoint files, so you pick the `.pptx` directly from where it lives.
4. Adjust the options you need:
 - **Image Format** and **PowerPoint Image PPI** control how rendered picture shapes are produced and how large the file is. PPI defaults to 200.
 - **Fit to Width** and **Fit to Height** set the deck-wide default for how an imported picture is sized against the template object it replaces. **Fit to Width** is on by default. Set one of the two, not both; see *Size a picture* below for what each one does.
 - **Sort slides to match worksheet order** reorders replicated slides to follow the worksheets they link to.
 - **Output PowerPoint as PDF** also renders the deck to PDF. The shared PDF options (table of contents, page numbers, low-quality mode) then apply.
5. Save the job.

Output formats are additive, so a single run can produce the deck, a PDF, and other formats together. See [Output formats](#).

Link the slides to report data

Links are authored in the **Reportworq add-in**, which loads a task pane in PowerPoint as well as in Excel. You link a shape in the pane's shape-link editor, and the link is saved on the shape itself.

Connect the deck to its job

Do this first. A shape link points at a report object inside a specific job, so the add-in has to know which job this deck belongs to before any link can be made. **The job must already exist in Reportworq**, saved, with its Excel reports added as sources. If you have not built it yet, do that before opening the template.

1. Open the template in PowerPoint and open the **Reportworq** task pane.
2. Sign in, if you are not already.
3. **Select the job** this deck reports on. It must be a job in the workspace you are signed in to.



LINKED JOB



Select Job

SLIDE SHAPES



Import new shape

Link a slide shape to a report object



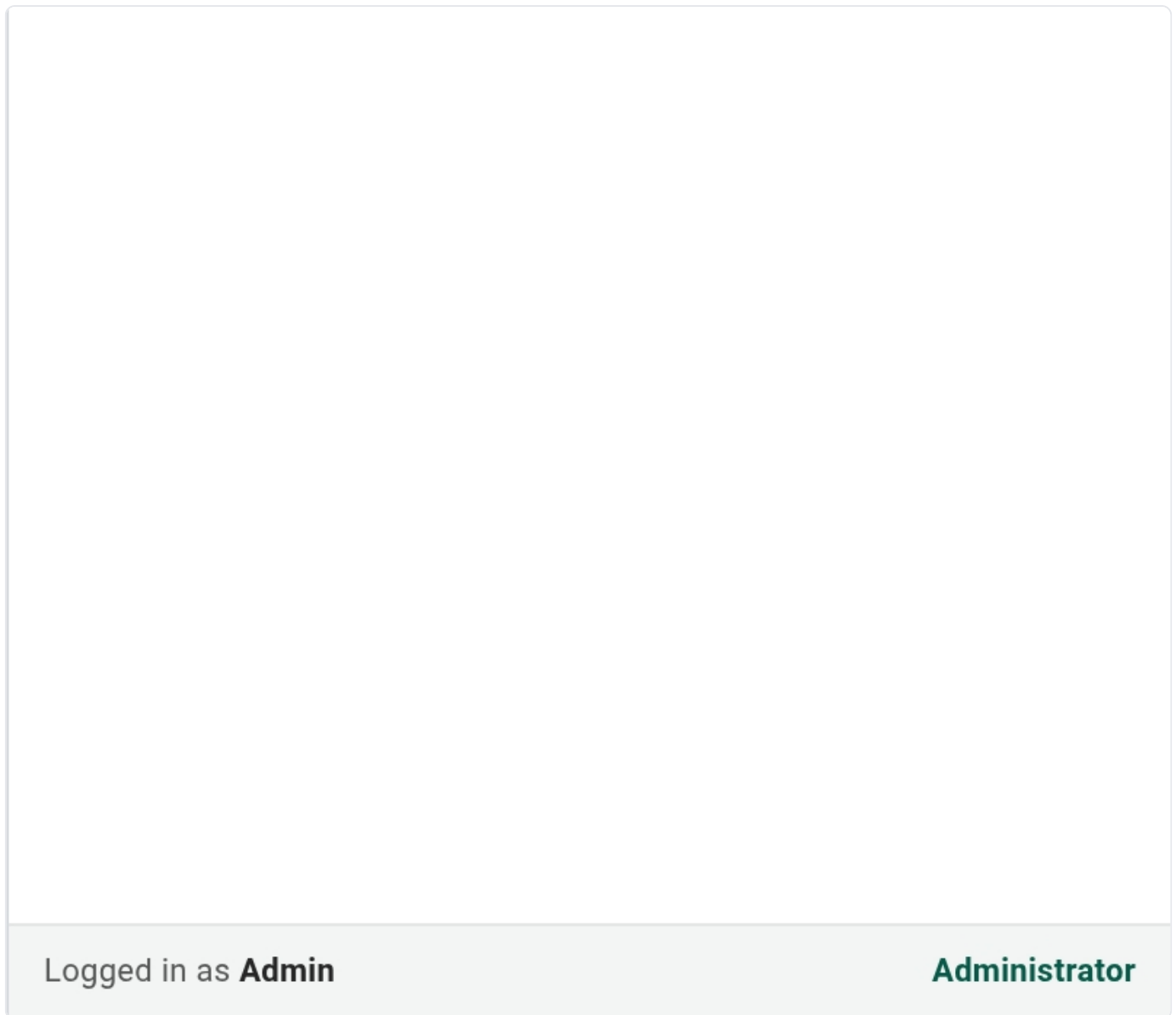


Unlinked text box

☒ Not linked



Select to link this shape to report data



Everything below assumes the deck is connected. If the job picker is empty, check that you are signed in to the right workspace and that the job has been saved.

Link a shape

The task pane's home view is **Manage shapes**: the connected job at the top, then a **Slide shapes** list where each shape carries a **Linked** or **Not linked** status chip.

1. On the slide, **select the shape you want to fill**, or choose the **Import new shape** card under **Slide shapes**. The editor header reads **Import linked shape** for a new or unlinked shape, or **Edit shape link** for one that is already linked.
2. Set the **Data link**. Choose **Set the data link** to open the picker, then drill through **Report > Sheet > Object**:
 - **Report** - choose the Excel report.
 - **Sheet** - choose the worksheet.

- **Object** - choose what to link: **Whole worksheet** (for an inline text box), a **chart**, a **named range**, a **text box**, or **Custom cell range**. Choosing **Custom cell range** reveals a box for an address such as `A1:C10`.
3. Choose the **Shape type**, which is how the data lands on the slide: **Picture**, **Data table**, **Inline text box**, or **Linked text box**. The available types depend on the object you chose. A chart can only be a Picture; a cell range or named range can be a Picture or a Data table; a text box becomes a Linked text box; the whole worksheet becomes an Inline text box. When only one type fits, the pane shows it as a single confirmed choice rather than a chooser.
 4. Set the **Options**, each a switch. Which ones appear depends on the shape type:
 - **Fit to width** and **Fit to height** appear for a cell-range or named-range **Picture**. See *Size a picture* below.
 - **Transparent background** appears for any Picture, including a chart.
 - **Copy source format** appears for a **Data table** or an **Inline text box**, and carries the source cell's bold and font color onto the slide.
 5. Choose **Import shape**.
 6. Repeat for the other shapes on the slide.

To link a shape you already placed, select it and set its data link the same way. This retrofits an existing corporate template in place rather than rebuilding it.

Tip. Use **Refresh Report List** (in the shape editor) or **Refresh Job List** (on the shape list) to reload the worksheet and object lists after you edit the Excel reports. Close the Excel reports first, so the refreshed lists reflect saved content.

Opening an older deck. Decks built before the add-in encode their links in each shape's Alternative Text. Reportworq still reads that format, so those decks keep working. Do not author new links that way; use the add-in.

Size a picture

A picture shape is rendered from the Excel range or chart it links to, then sized against the template object it replaces. **Fit to width** and **Fit to height** decide how.

Set **one of the two, not both**:

Setting	The shape on the slide	The picture inside it	Aspect ratio
Fit to width only	Keeps its width, and its height is recalculated to suit the picture.	Sized to the shape's width.	Preserved
Fit to height only	Keeps its height, and its width is recalculated to suit the picture.	Sized to the shape's height.	Preserved
Both	Does not resize at all. The shape stays exactly the size you drew it.	Stretched to fill the shape's exact width and height.	Not preserved. The picture can look distorted.
Neither	Same as Fit to width only.	The job's deck-wide Fit to Width and Fit to Height defaults apply, and if the job sets neither, the picture is fitted to width.	Preserved

So the choice is which dimension you want to control, and whether you are willing to let the shape move. Pick **Fit to width** when the shape's width has to line up with the rest of the slide, and accept that its height will change. Pick **Fit to height** when the shape sits in a fixed-height band and its width can vary.

Turning **both** on is a different kind of choice, and it is the one that surprises people. It is the only setting that leaves your shape exactly where and as you drew it, which sounds like what you want, but it gets there by stretching the picture to fit rather than by resizing the shape to suit the picture. Use it only when the layout genuinely cannot move and a little distortion is acceptable. If a slide looks squashed or stretched, this is the first setting to check.

Shape settings override the job's defaults, so a deck can fit to width everywhere except the one wide table that needs to fit to height.

Show a value inside a sentence

An **Inline text box** links to the worksheet as a whole. Choose **Whole worksheet** on the Object step of the data link, and mix literal text with variables in curly braces, for example `Revenue grew to {Revenue_Total} in {Region}`. At run time each variable is replaced with its cell or named-range value. With **Copy source format** on, the substituted figure also takes the source cell's bold and font color. Use **Toggle variables** to switch between the tokens and the substituted values while authoring.

If a variable names a range that does not exist, the slide shows `RANGE '<name>' NOT FOUND` instead of a value.

Test a slide before you save

While authoring the source template, the pane gives you per-slide and per-shape actions:

- **Refresh slide** repopulates every linked shape on the active slide from the report.
- **Clear slide** replaces linked data with placeholders, which is useful for finding which shapes are live.
- **Refresh data** and **Clear data** do the same for a single shape, in its **Actions** section.
- **Reset shape** reverts a picture to its original size.

How slides replicate when you burst

When a parameter fans a source worksheet into several copies, each slide linked to that worksheet **replicates into one slide per item**, carrying that item's data. **Adjacent slides that share the same replicating worksheet replicate together as a group**, repeated in sequence, so each recipient gets their block of slides.

A single slide may draw from **more than one** worksheet, but **only one of those worksheets may be the replicating one**. If a slide links two replicating worksheets, generation fails rather than producing an ambiguous deck.

A PowerPoint report bursts and distributes through the same pipeline as an Excel report: one deck per burst slice, to the configured destinations. See [Replicate output from your source system](#).

Let recipients refresh or run the deck (optional)

Refreshing or re-running a delivered deck happens in the PowerPoint add-in and requires a signed-in Reportworq account. A plain recipient who only receives the deck needs nothing. Two permission gates in the PowerPoint options control what is possible:

- **Allow PowerPoint user to refresh presentation output** lets a recipient pick a job-history version and refresh the deck from it.
- **Allow PowerPoint user to run Job** governs the on-demand run affordance. It is available only when **Allow ... refresh** is also on, and it clears automatically if you turn refresh off.

The two options do different things, and the difference is worth being clear about:

- **Refresh** rebuilds the deck from a **run that already happened**. The recipient picks a version from the job's history and the slides are repopulated from that run's output. Nothing is re-queried.

- **Run Job** starts the Reportworq job. The job goes and gets fresh data the way any run does, and the recipient then sees the output of that new run. **Run is offered only to an Author or an Administrator.** A recipient holding a **User** license never sees it, even when the job option is on.

So a recipient never reaches into your data source from PowerPoint. They either re-render a stored run or, if they are an Author or Administrator, ask Reportworq to run the job.

In the add-in, someone opening a delivered deck can:

- **Select Job History** and pick which generated version to refresh from.
- **Refresh slide** or **Refresh presentation** to repopulate from that version.
- **Download** the deck as Reportworq generated it, without their own local edits.

What appears in that history list is governed by the same permissions as the rest of Reportworq: a person sees history only for a job they are allowed to view, and only the output variations their secured lists allow.

Refreshing **always loads from a job-history version, never the live data source in real time.** To see current data, someone must run the job to produce a new history record. Each run emits a **PowerPoint Datasource.xlsx** companion file alongside the deck, and that file is what powers the refresh actions.

Notes and limits

- **A template is required.** There is no headless PowerPoint output without one.
- **One replicating worksheet per slide.** A slide may source several worksheets, but only one may be the fanned-out (replicating) one, or generation fails.
- **Refresh is always from job history, not live data.** Both author-side and recipient-side refresh read a saved version.
- **Deleting a job-history entry removes its refresh data.** If a history version, or its PowerPoint Datasource.xlsx, no longer exists, the refresh actions for it disappear.
- **Inline text boxes** are re-evaluated only on a full run, or through **Toggle variables.** **Refresh slide** and **Clear slide** do not touch them.
- **Fit to width and Fit to height apply to pictures built from a cell range or a named range.** A chart picture offers only **Transparent background.**
- The Report Viewer presents PowerPoint output through a **Slides** thumbnail rail rather than an inline document view. See [View and download results](#).

Going deeper. A script stage runs after the deck is generated and before it is delivered, with the whole presentation available to it. Use it to stamp footers and slide numbers, remove draft slides, or replace text tokens on the slides. See [The Script Runner](#).

Related topics

- [Choose the source workbook and worksheets](#)
- [Output formats](#)
- [The Office add-ins](#)
- [The Script Runner](#)
- [Replicate output from your source system](#)
- [View and download results](#)

Charts and maps

explanation

report author

A Reportworq report can carry two kinds of visual, and they are not in competition. Most reports end up using both.

- **Native Excel charts.** The source workbook is a real Excel file, so any chart you build in Excel is supported as it stands.
- **Reportworq charts and maps.** The Excel add-in adds its own chart and map builder on top, for the visuals Excel's own charts do not cover and for visuals that have to redraw themselves on every run.

Native Excel charts

Nothing about a Reportworq report asks you to give up Excel charting. Build the chart in Excel the way you always have, and it goes on working:

- **It plots the cells Reportworq populates.** A scheduled run recalculates the workbook, so the chart is drawn over that run's numbers.
- **It can fill a PowerPoint slide.** A slide shape can be linked to a chart in the Excel report by name, and Reportworq renders it into the deck on every run. See [PowerPoint reporting](#).
- **It can go into an email body.** The **Chart** variable inserts a chart from the workbook into a message template, and it can take the chart's name from a cell so the run decides which chart to show. See [Message templates](#).

If Excel already draws what you need, use an Excel chart. There is no conversion step and no penalty for it.

What the add-in builder adds

Reach for the builder when Excel's own charts do not cover what the report needs, or when a visual has to keep itself current through scheduled distribution. It offers the following capabilities.

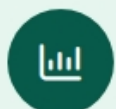
Capability	What it means in practice
12 chart types, plus maps	Column, bar, line, stacked column and stacked bar, pie with a center-hole slider that makes it a donut, waterfall, combo, range bands drawn as bars or as an area, heatmap, and Sankey.
Shapes built for a finance review	A waterfall cumulative bridge for a variance walk, a Sankey flow diagram whose ribbon widths carry the values, a heatmap that colors a matrix around a neutral point you choose, range bands that float from a start value to an end value instead of rising from the axis, and a combo chart that puts a rate on a second axis alongside amounts.
Maps over tiles or over your own image	Labeled points and value-sized bubbles drawn over real OpenStreetMap tiles, or over an image you supply. The image does not have to be a map: a store floor plan, a site layout, or a schematic works the same way, with your data plotted onto it.
Maps driven by your own data	The place the map centers on, the place name it searches for, and the zoom level can each read a worksheet cell instead of a fixed value. One authored map then serves a whole burst set, framing itself on each recipient's own location.
Redraw with nobody opening Excel	Every Reportworq chart and map on the workbook is redrawn from current cell values when the job runs.

ON THIS WORKBOOK



New report

Connect to data and import a list



New chart

Build a chart or map from a range and import it



Nothing on this workbook yet

Start with New report or New chart to add content to

this workbook

THIS WORKBOOK.



Formula library

20 capabilities · Reporting · Contribution · Data



Interactive charts and static visuals

The one real difference between the two is worth stating plainly, because it is what decides which you reach for.

An Excel chart is interactive and automatic inside Excel. It reacts as you work: change a cell it plots and it redraws immediately, and its data points respond when you point at them.

A Reportworq chart or map is a static picture. It is a visualization, not a live object. It does not respond to pointing or selection, and it does not redraw while you edit the sheet. What it does instead is carry its own definition, so it can be redrawn on demand:

- **Refresh** in the add-in redraws a chart, or every chart on the workbook, from the current worksheet values.
- **A job run redraws all of them automatically**, from that run's data, with no one opening Excel.

So the trade is interactivity inside the sheet, in exchange for a visual that maintains itself through scheduled distribution and comes in shapes Excel's own chart types do not cover.

Which one to use

Use a **native Excel chart** when Excel already draws the shape you want and you value the live behavior in the sheet.

Use the **Reportworq builder** when one of these is true:

- You need a **map**, or a chart shape Excel's own types do not cover.
- The visual belongs to a **burst set**, and each recipient's copy has to reframe itself on their own region, product, or entity.
- You want the visual defined once and redrawn by every scheduled run, rather than rebuilt by hand.

Going deeper. Every chart type and the data shape it expects, the full mapping capability including cell-driven framing and the zoom lookup technique, and how refresh works, are in [Excel: charts and maps](#).

Create an AI Profile for a report

[how-to](#)[report author](#)

The **AI Profile** step of the Job Editor lets you attach a plain-language description and audience metadata to a report so that AI assistants, search, and recommendations know **when the report should be referenced**. It is metadata only: it changes how the report is discovered and surfaced, not how the report renders or what it outputs.

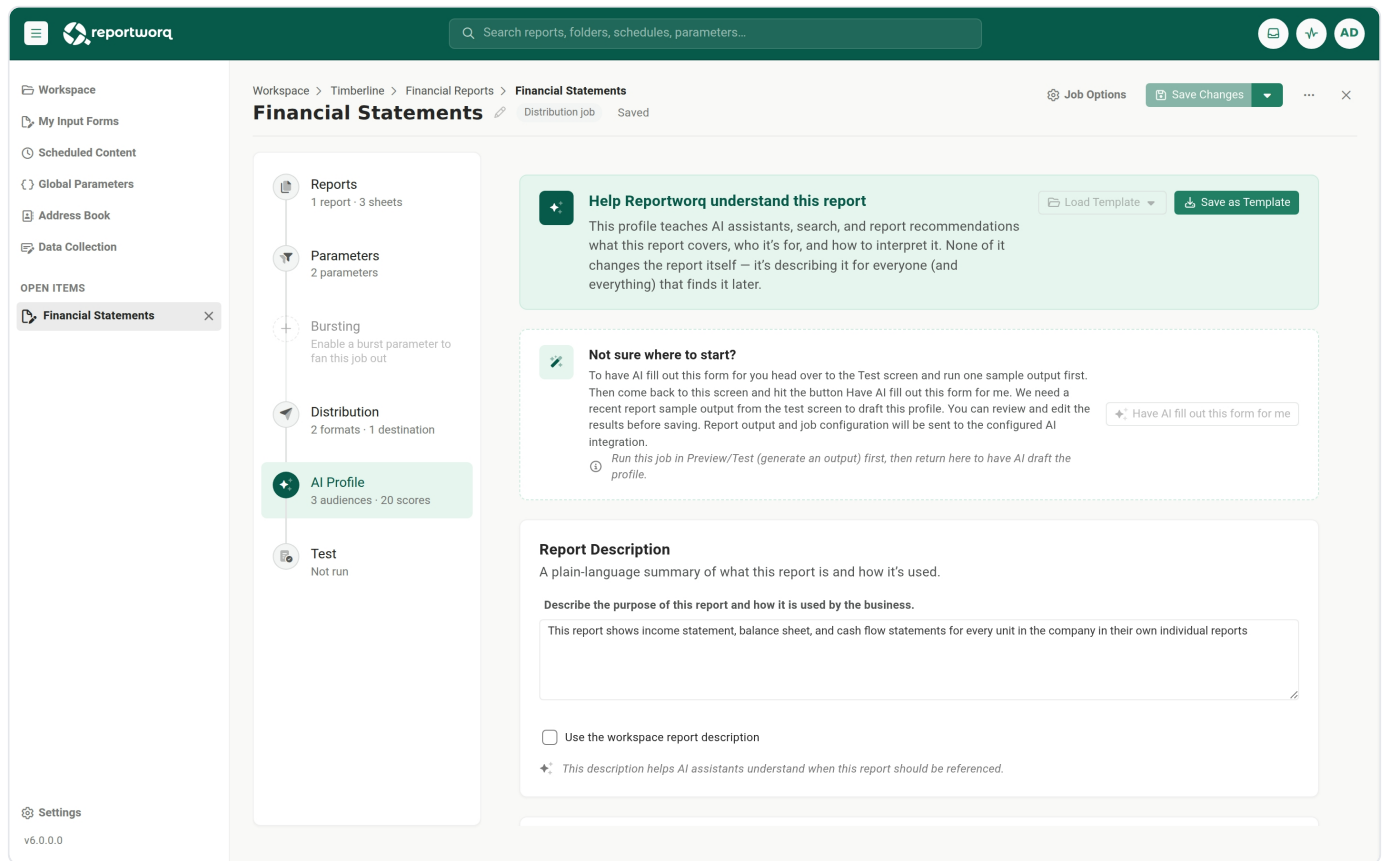
This is Reportworq's "AI done differently" idea. Most AI systems try to reverse-engineer a business from raw databases. Reportworq instead presents AI with reports your finance teams have already curated, and the AI Profile is where you enrich a report with the structured business context, purpose, audience, strengths, limitations, terminology, and caveats, that teaches AI how your business already communicates. The result is lower token usage, greater confidence, and answers based on the same information your teams already trust.

When to use it. Fill in an AI Profile once a report is functionally complete, especially in a large report library where people (and assistants) need help finding the right report. A well-described report is the one an assistant surfaces when someone asks a natural-language question.

Before you start

- A report to describe. The AI Profile is the fifth Job Editor step, before **Test**.
- The value is realized wherever AI, search, and assistant surfaces are in use. See [AI in Reportworq](#).
- Writing the profile by hand needs nothing else. To have AI draft it instead, an AI connection has to be configured for your instance and the report has to have generated a sample output. See [Speed it up with AI prefill and templates](#).

Describe the report



1. Open the report and go to the **AI Profile** step.
2. Write a **Description** of what the report contains, what questions it answers, and when it is relevant. Aim for the questions a reader would actually ask. Alternatively, turn on **Use workspace description** to inherit the workspace's description instead of writing a per-report one.

Where descriptions come from. Every workspace item, a report, a folder, a data model, or a campaign, carries its own free-text **Description** on its Properties panel, and that is what the workspace description above refers to. Descriptions are worth filling in for their own sake: they show in the item table and in search results. See [Item properties](#).

3. Set the **Audience** and **Data Type** tags. Pick from the existing options or create new ones inline, so the lists grow with your library.
4. Set the **Relevance scores**: for each business-dimension category, drag the 0-10 slider to say how strongly this report relates to that dimension. You can edit the categories themselves with **Edit Categories**.

5. Save the job.

Speed it up with AI prefill and templates

Reportworq can draft the whole profile for you. Two things have to be in place first, in this order, and neither is optional: an AI connection, then a sample output from the report itself.

Set up an AI connection

AI prefill uses your organization's own AI provider, not a model built into the product, so nothing AI-assisted works until a connection exists. An administrator creates it in **Settings ► Integrations**, in the **AI Connections** group: select the provider tile (OpenAI, Azure OpenAI, Amazon Bedrock, or Reportworq Hosted AI, depending on what your instance offers), enter its credentials and model, and turn on **Enable provider**. Use **Test connection** in the editor before saving, so a wrong key, endpoint, or model surfaces there rather than as a failed draft later. See [Connect an AI provider](#).

Until a connection exists, the AI prefill panel does not appear on the AI Profile step at all. If you cannot find the button, that is the reason, and it is an administrator's task rather than yours.

Generate a sample output

AI drafts the profile from the report's actual content, not from its definition alone, so the report has to have run and produced an output for AI to read.

1. Go to the **Test** step of the Job Editor.
2. Select **Generate Output** and let the run finish successfully.
3. Return to the **AI Profile** step.

Until that run has produced an output file, **Have AI fill out this form for me** stays disabled, with the note *"Run this job in Preview/Test (generate an output) first, then return here to have AI draft the profile."* Not every output format can be read for a sample. If the format is not supported, Reportworq asks you to generate an Excel output and try again.

Draft the profile

1. On the **AI Profile** step, select **Have AI fill out this form for me**.
2. If the profile already holds values, confirm the prompt to replace them. AI prefill overwrites the fields it fills.
3. Review every drafted field and correct anything wrong. The draft is a starting point, not an answer.

4. Save the job. Nothing is saved automatically.

The report's output and its job configuration are sent to the configured AI connection to produce the draft. Treat that the way you treat any other data leaving your instance, and confirm the report's content is appropriate to send before using prefill on it.

Reuse a profile with templates

Use **Save as Template** to keep a finished profile, and **Load Template** to apply it to a similar report. Templates need no AI connection. **Load Template** is unavailable until at least one template has been saved.

Notes and limits

- The AI Profile is discovery metadata. It does not change the report's data, worksheets, parameters, or output.
- Audiences, data types, and scoring categories are author-chosen and extensible, not a fixed system list.
- The profile feeds AI-facing surfaces: the in-app assistant, AI and global search, recommendations, and Copilot or MCP access to reports.
- AI prefill needs both a configured AI connection and a successful Test-step output. It overwrites the fields it fills and never saves on your behalf, so review the draft and save the job yourself.

Going deeper. How governed AI works across Reportworq, which providers you can connect, and how assistants, Copilot, and MCP consume report context is covered in the [AI & Integrations guide](#).