reportworq

Timberline Demo System

Enterprise reporting, distribution, and governed AI - product documentation

Version 6.0.0.0

Contents

Timberline Demo System

- Download the demo system 4
- Timberline release notes 6
- Start here 7
- What Timberline is 10
- The report catalog 13
- The campaign catalog 19
- Check out the AI features 22
- How Timberline rolls the reporting year forward 25
- Signing in, and who to be 30
- How parameters work 33
- Build a campaign 38
- Use the MCP server 42
- Where the files live 45
- What bursting looks like 49
- Update an input form 52
- Use a global parameter 55
- Export campaign data 57
- Build a compelling email dashboard 60
- Campaigns at large scale 64
- Try out PowerPoint 67
- Schedule jobs 70

Timberline Demo System

TIMBERLINE DEMO SYSTEM · ORIENTATION

Download the demo system

how-to | report author | administrator

Everything described in this guide is available as a **content store you can point your own Reportworq at**. It carries the reports, jobs, campaigns, schedules, parameters, contacts and data that the rest of this guide walks through.

⬇ Download the Timberline demo system (ZIP)

The download band on the documentation home page always carries the current version, date and size. What is on this page is the same package.

What you need

Reportworq 6, and nothing else. You do **not** need to connect a data source. The data the reports read travels inside the package, so the numbers are real and the reports run anywhere.

The fastest way: one click in the setup wizard

A **brand-new Reportworq install ships with this demo built in.** On first launch, the setup wizard's "Where should Reportworq store its data?" step offers **Set up the Timberline demo system**. Choose it and Reportworq extracts the demo into a `TimberlineRepo` folder next to where a normal repository would go, points itself at it, and restarts to the sign-in page - no download, no unzip, no path to type. The wizard shows you the sign-in (`admin@rw.com` / `Admin_1234`) before it restarts and again on the way in.

There is no license in the demo, so Reportworq asks you to activate with your own key on first sign-in, exactly as any fresh install does. Prefer this route on a new install; use the download below to add the demo to an install you have already set up, or to get a newer version than the one bundled with yours.

Install it (download route)

1. **Extract the ZIP.** You get a single folder, `Timberline-Demo-System`.
2. **Put it where Reportworq can read and write it.** Alongside your existing repository is fine; it is a self-contained content store, not something that merges into one you already have.
3. **Point Reportworq at it** as the repository, and start the server.
4. **Sign in** as `admin@rw.com` with the password `Admin_1234`, then change it.

There is no license in the package. Reportworq will ask you to activate with your own key exactly as a fresh install does.

> **Treat it as a sandbox, not a starting point.** Explore it, run everything, break things. When you want to build your own content, start a clean repository and bring the *patterns* across rather than editing Timberline into your production system. Where the files live explains why the jobs are more fragile to renaming than they look.

What is inside

Reports	Financial statements, board packs (Excel and PowerPoint), sales information, commission statements, rep scorecards, chart examples
Jobs	15, including bursts by market and by burst set, a scheduled job, a parameter-override job, and one deliberately-failing job to show what a failure looks like
Campaigns	A budget input campaign with contributors and approvers already assigned
Data	A SQLite database and a journal workbook, both inside the package, wired up through the repository's own file area
Accounts	Ten, from workspace admin down to read-only, so you can see the product as different people

Parameters still work

Where a report offers a parameter, a market or a year, it behaves exactly as the guide describes: the parameter drives a lookup into the packaged data, so a burst by market produces one genuinely different output per market rather than the same file under fifteen names.

Release notes

Changes to the demo system are recorded in the release notes.

Timberline release notes

reference · report author · administrator

The downloadable Timberline demo system is published on its own cadence, separate from Reportworq releases. It is content, not product, so it can be corrected and extended without waiting for a server release.

Changes to the demo system will be recorded here as they are published. The download band on the documentation home page always carries the current version and date.

TIMBERLINE DEMO SYSTEM · ORIENTATION

Start here

explanation report author administrator

Timberline Harvest Co. is Reportworq's own worked example: a complete, working deployment built on one fictitious company, used for demos, enablement, testing and marketing. If you want to see what Reportworq actually does rather than read about it, this is where you look.

Start here: the order that makes sense

If you are opening Timberline for the first time, go in this order. Each step builds on the last.

1. **What Timberline is**: the company, the four layers, and what is actually in the workspace. Ten minutes.
2. **Signing in, and who to be**: the ten accounts, what each one sees, and how to read what a recipient received without signing in at all.
3. **Where the files live**: the workbooks behind every report, and the one rule that keeps you from breaking a job.
4. **Open the Financial Statements job first**, ahead of the more eye-catching board decks. It is the most straightforward example of how a source file and a job fit together, and everything else in the catalog is a variation on that shape. See it in the report catalog.
5. Then read the rest of the report catalog and follow whichever task guide matches what you are trying to do.

Run reports

I want to…	Read
See what I can run, and run it	The report catalog
Understand how one template becomes many outputs	How parameters work
See a burst actually fan out	What bursting looks like
Change the reporting year once, for everything	Use a global parameter
Build an email people actually read	Build a compelling email dashboard
Show Excel to PowerPoint	Try out PowerPoint
Put a job on a clock	Schedule jobs

Collect data back

I want to…	Read
See what campaigns exist and what each is for	The campaign catalog
Build a campaign from scratch	Build a campaign
Change what contributors type into	Update an input form
Get the collected numbers back into the model	Export campaign data
Run a campaign with many forms and many people	Campaigns at large scale

AI and integrations

I want to…	Read
Show the AI features	Check out the AI features
Answer questions over Timberline from an AI client	Use the MCP server

What people always miss

Worth reading before anything else. Two mistakes come up again and again:

1. **Setting up parameters the wrong way, or on the wrong sheet.** A parameter maps to one cell *per worksheet*, and getting the mapping right per tab is where people go wrong. See How parameters work.
2. **Hand-maintaining recipients instead of automating them.** In her words: pull the contact information into your source file and use cell values or named ranges to get the **Email To** and **CC** right, dynamically sourced from your source system where possible, rather than controlling it manually in Reportworq via the bursting screen. Every Timberline job does it this way. See What bursting looks like.

And three more that catch people out:

3. **The email is the deliverable.** Timberline's jobs do not send "please find attached", they build a readable dashboard in the message body. Open the demo inbox before you open an attachment.
4. **Nothing escapes the box.** Email runs in demo-inbox mode, so a demo run cannot mail a real person. Check that before you run anything on a shared environment.
5. **Jobs bind to named ranges, not cells.** Rename a range in a workbook and the job breaks at run time, silently. See Where the files live.

6. **The three board-deck jobs are deliberately similar.** They differ in *replication*, and reading them side by side is the fastest way to understand parameters.

A note on what you are reading

Timberline is a *worked example*, not a product feature. Everything in it was built with the same screens you have. Where this guide says "the demo does X", you can do X the same way in your own workspace, and the point of each page is the transferable pattern rather than the demo itself.

What Timberline is

explanation | report author | administrator

Timberline Harvest Co. makes and distributes woodworking products (*Timber & Tools*, about 60% of revenue) and agricultural and homestead equipment (*Farm & Field*, about 40%). FY2026 net revenue is about $820M, across 3 regions and 13 markets, 4 sales channels, and a 237-person sales organization. There are actuals for 2022 through 2026 and a 2027 Plan and Forecast.

None of it is real. All of it reconciles.

Four layers

Timberline is not one artifact. It is four, stacked:

1. The dataset. A deterministic Python pipeline generates every number in the company from one configuration file with a fixed random seed. Rerun it and you get byte-identical output, which is why a screenshot taken last month still matches the numbers on screen today. It also *verifies* itself: 12 assertions check that the sales journal sums to the sales totals, that the commission journal traces to individual invoices, that the balance sheet balances at every month end, and that cash flow reconciles to the cash movement.

2. The planning model. The generated numbers load into a planning model: 7 cubes (sales, income statement, balance sheet, cash flow, sales ops, compensation, and a control cube) over 15 dimensions.

3. The content. Workbooks and decks that read that model: the board pack, the market review, the financial board book, the sales information workbook, the rep scorecard, the commission statement, and two contribution form templates, plus a set of parameterized templates built for AI to answer questions from.

4. The application. A complete Reportworq v6 workspace: 8 jobs, 2 contribution campaigns, 10 user accounts in 8 groups, 2 data connections, a schedule, message templates, run history, and an inbox full of mail that previous demo runs actually sent.

Layer 4 is what people mean when they say "open Timberline". Layers 1 to 3 are what make it defensible.

What is actually in the workspace

Everything lives in one workspace (`Default`), in one folder tree:

```
Timberline
└─ Financial Reports          ← the 8 jobs and 2 campaigns live here
   └─ Source Files            ← the 9 workbooks and decks live here
```

Eight jobs:

- Board Packet · Board Packet with Division Slides · Divisional Packet, three ways to build a board deck, differing in how they replicate
- Financial Statements · Financial Statements - SharePoint: the same statements, delivered one way and four ways
- Timberline Sales Information: the deep sales analysis, and the AI showcase
- Sales Rep Scorecard: manager-and-team fan-out, with generated commentary in the email
- Commission Statements: bursting at scale, rolled up into one email per manager

A contribution campaign:

- Segment & Channel Budget Input: the full planning cycle: forms, approvals, write-back to two cubes

See the report catalog and the campaign catalog for what each one is for.

Why it is built this way

Three design decisions explain most of what you will see.

Everything ties. The board deck, the sales report, the commission statement and the budget form all read the same model. A number quoted on one screen is the same number on the next. That is what lets a demo move around without contradicting itself, and it is the reason the pipeline exists.

The model owns the metadata, not Reportworq. Recipient email addresses, manager reporting lines, region-to-VP mappings and the current reporting year all live in the planning model as dimension attributes and control values. The workbooks look them up. So the distribution list is never stale, and rolling the year forward is one change in one place.

Delivery is part of the report. The jobs build real email dashboards: a greeting, a headline number, a picture of the report, a live HTML table, and on two jobs a generated commentary paragraph, with the full detail attached for whoever wants it. Judging Timberline by its attachments misses the point.

The safety property

Email is configured in **demo-inbox mode**. Delivered messages land in an inbox inside the application instead of going to a mail server, so a demo run cannot mail a real person. Everything else follows from that: you can run any job, any number of times, and read exactly what each recipient would have received.

Check that setting before running anything on an environment you did not set up yourself.

Related

- Signing in, and who to be
- Where the files live

The report catalog

reference report author

Eight jobs live in **Workspace ▸ Timberline ▸ Financial Reports**. They are not variations on one idea, each was built to show a different part of the product.

The eight

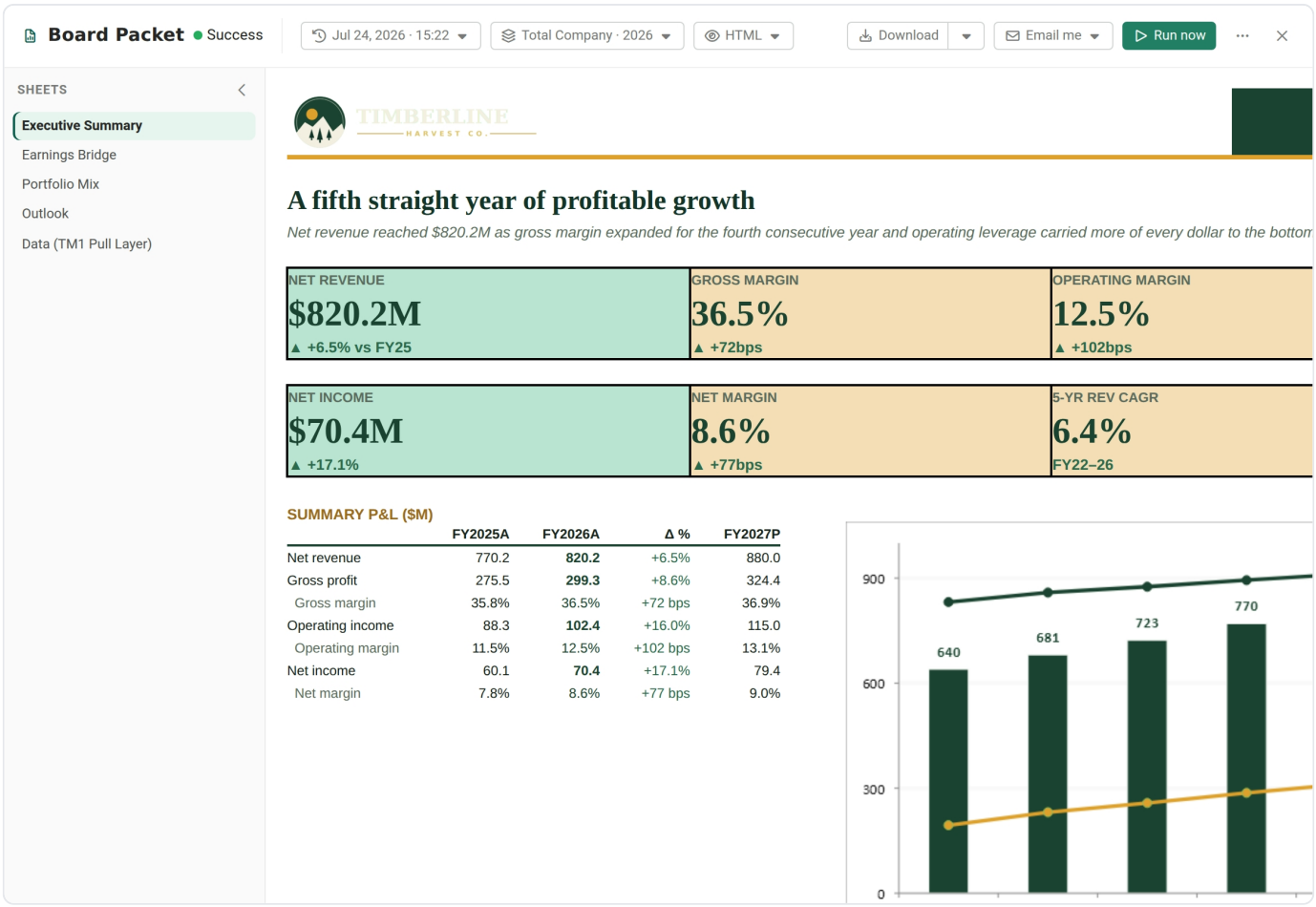
Board Packet

Executive board deck for total company and regions.

Produces a **PowerPoint deck per region**, four decks, one each for Total Company, North America, EMEA and APAC. The deck is built from `Timberline_Board_Pack.pptx`, whose slides are linked back to named ranges and charts in the workbook, so nobody edits PowerPoint.

The email is the other half: a greeting pulled from the workbook, a 50%-scaled screenshot of the KPI band, and the summary P&L rendered as a live HTML table.

Read this one when you want Excel to PowerPoint, or an email people actually read.



Board Packet with Division Slides

One high-level deck, with a division section repeated inside it.

This job exists to show replication in PowerPoint. Its second source workbook contributes a single Revenue Review tab, which repeats per market, and the slide section bound to it repeats with it. That is the only reason the extra workbook is there.

Same content as Board Packet, different mechanics. The pack stays at **Total Company** level and `Market` is set to **one page per item** across four values, so instead of four decks you get **one** deck whose sales revenue slide repeats once per division. Outputs Excel, PDF **and** PowerPoint.

Read this one when you want the difference between "a file per thing" and "a page per thing".

Divisional Packet

A board deck per division, each containing only that division's markets.

The most interesting job in the workspace, and the counterpart to the one above. Where the division-slides deck stays at total-company level and repeats divisions **inside one deck**, this one runs **per division** and then repeats each market **under that division** inside the pack it produces.

Both parameters are MDX, and the second depends on the first:

- `Region` drills one level down from Total Company: so one output per region.
- `Market` drills down from **whichever region this output is**: so each divisional pack contains exactly its own markets.

Nothing is hard-coded. Add a region to the planning model and a new pack appears. Emails are combined by region, and the PDF gets a table-of-contents title built from the region name.

Read this one when you are wondering whether you will have to maintain a list. You will not. Read it **next to Board Packet with Division Slides** when you want the two replication modes side by side, which is the fastest way to understand them.

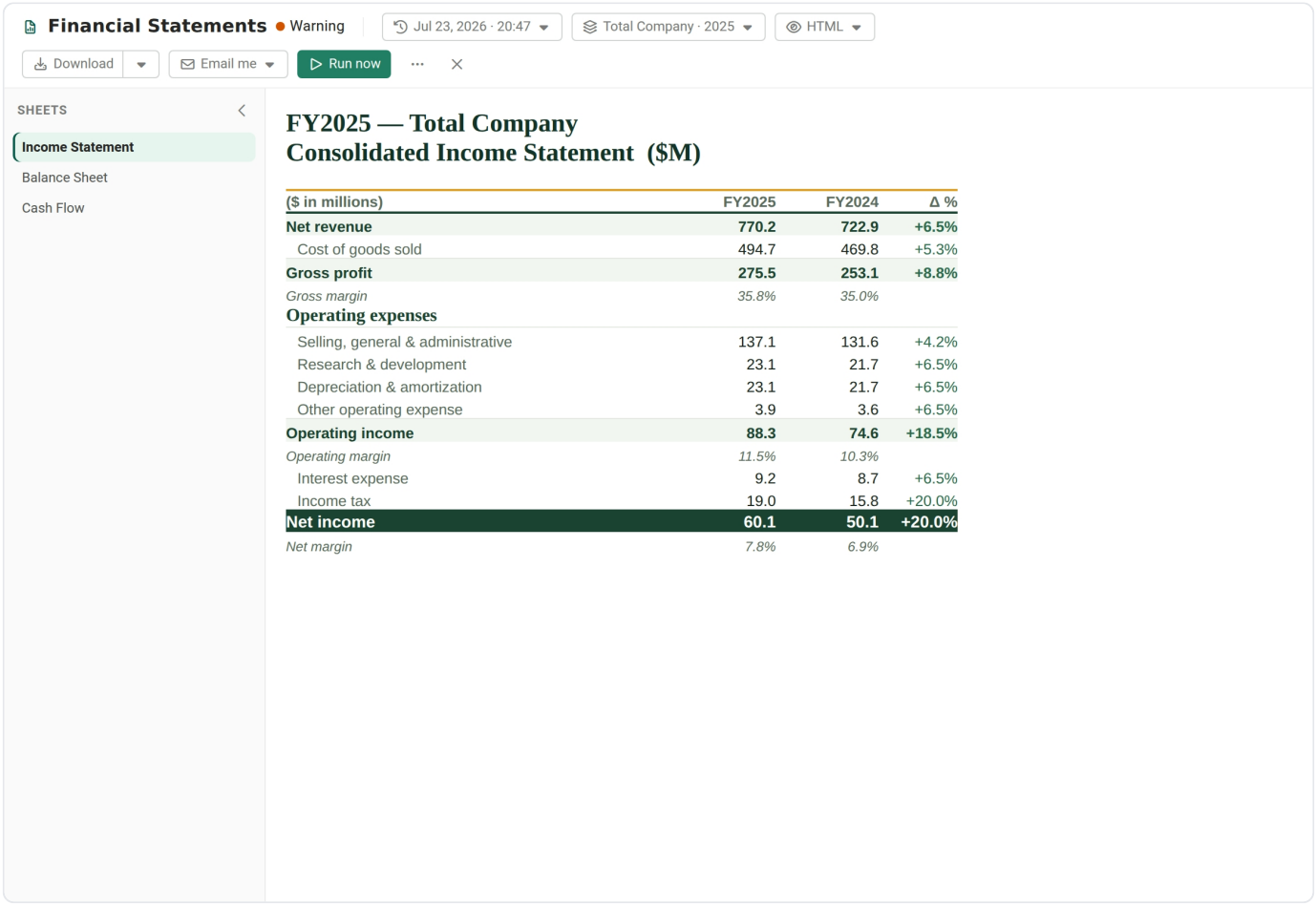
Financial Statements

Income statement, balance sheet and cash flow, per market.

Start here if you are new. This is the most straightforward example of how a source file and a job fit together. Learn the shape here, then read the board decks.

A classic per-entity burst. `Market` comes from a subset in the planning model, `Year` from the **Current Year** global parameter. Outputs Excel and PDF, with a cover page, page numbers, and a table-of-contents title carrying the market name.

Read this one when the audience is finance and wants to see the bread-and-butter case.



Financial Statements - SharePoint

The same statements, delivered four ways at once.

Identical content, four destinations: **Email**, **SharePoint**, **OneDrive** and **Google Sheets**. The interesting part is the paths, the SharePoint and OneDrive folders are built from parameter values:

```
…/Shared Documents/Timberline/{Market}/{Year}
```

So the output files itself by market and by year, and the folder tree is created as it goes. Nobody pre-creates folders.

Read this one when someone says "we need it in SharePoint too". *(The targets are Reportworq-internal; demo from configuration unless the tenant is available.)*

Timberline Sales Information

Sales by product, channel, market and the product-by-channel cross-tab, monthly.

The deepest report and the AI showcase. Four parameters, `Market` produces one report each, while `Channel` and `Product` each produce a page, so a single template yields a tab per channel *and* a tab per product, with the tab names built from the values. Empty rows suppress themselves.

It carries an **AI Context** worksheet and a full AI profile, and its email body contains a generated commentary paragraph rather than a static message.

Read this one when the conversation turns to AI, or to "how deep can one report go".

Sales Rep Scorecard

Team performance for a manager, plus a page per direct report.

Manager-and-their-team fan-out from one MDX pattern: `Manager` filters the employee dimension by role, then `Rep` filters it by *this* manager's id. One PDF per manager, one page per rep inside it. The roster is the model's roster, so it never goes stale.

The email opens with the manager's own attainment figure, then a generated paragraph naming the top and bottom three reps in a management-review voice.

Read this one when someone wants to see AI commentary that is actually about *their* numbers.

Commission Statements

A statement per rep, delivered as one email per manager.

Bursting at scale. `RepID` produces one statement per rep and `Period` produces a page per period, but nobody wants nine separate emails, so the job **combines** on a `Territory` cell value. Each manager receives one message containing their whole team's statements.

The source workbook lives on the **Network Folders** provider, not in workspace storage, so this job needs the demo host's network root.

Read this one when the objection is "we would drown people in email".

How to run one

1. Open **Workspace ▸ Timberline ▸ Financial Reports** and select the job.
2. Use **Test / preview** first, it shows you what the job will produce without delivering anything.
3. To run for real, use **Run** from the job or from the Run Jobs screen. Choose **Hold for delivery** if you want the output produced but held for a human to release; otherwise it delivers immediately to the demo inbox.
4. Watch it in **Activity**, then read the result in **Report Viewer**, or open the **demo inbox** and read the mail as the recipient did.

> The exact screen names above follow the customer User Guide; this guide has not yet been re-checked click-by-click against the live demo build.

At a glance

Job	Fans out by	Outputs	Destinations	AI
Board Packet	Region (report each)	PowerPoint	Email	profile
Board Packet with Division Slides	Market (page each)	Excel, PDF, PowerPoint	Email	profile
Divisional Packet	Region (report) × Market (page, cascading)	Excel, PDF	Email	,
Financial Statements	Market (report each)	Excel, PDF	Email	profile
Financial Statements - SharePoint	Market (report each)	Excel, PDF	Email, SharePoint, OneDrive, Sheets	profile
Timberline Sales Information	Market (report) × Channel, Product (pages)	Excel, PDF	Email	profile + insight
Sales Rep Scorecard	Manager (report) × Rep (page, cascading)	PDF	Email	profile + insight
Commission Statements	RepID (report) × Period (page)	PDF	Email, combined per manager	,

Related

- How parameters work · What bursting looks like
- Build a compelling email dashboard · Try out PowerPoint

TIMBERLINE DEMO SYSTEM · CAMPAIGNS

The campaign catalog

reference report author

A **campaign** is a job whose destination is data collection: instead of emailing output, it generates **input forms**, assigns them to people, collects their numbers, and writes the results back to the planning model.

Timberline carries one, and it exercises nearly every contribution capability at once.

Segment & Channel Budget Input: the full cycle

The reference campaign. It exercises nearly every contribution capability at once: form generation, assignment, validation, approvals, notifications, and write-back to two different cubes.

Three forms, one per market, USA - East, USA - Central and Southeast Asia, generated from `Timberline_Budget_Input_Model.xlsx` and named from the market value. The year comes from the **Current Year** global parameter, so the whole campaign re-points to a new plan year with one change.

The form has three sheets:

- **Budget Entry**: the input grid. Twenty-four input cells across product lines and channels, each validated so a non-numeric entry is rejected with *"Not a number"*. Volume, price/mix and variable-cost drivers on one binding; five row-blocks bound to the plan model.
- **Fixed Costs**: a smaller input block on its own model.
- **Summary**: a read-only contribution-margin waterfall that recalculates as the contributor types.

Who is assigned:

Form	Contributor	Approver
USA - East	the USA - East business seat	the North America seat
USA - Central	the USA - Central business seat	the North America seat
Southeast Asia	the APAC business seat	the Southeast Asia seat

Note the shape rather than the names: contributors sit in the region that owns the number, and one approver covers more than one form.

Options in force: approvals required, approvers may edit forms, export/import enabled, exported workbooks hide the non-print area and lock non-input cells, invitations sent from a nominated sender address.

Ten automation tasks: invitations on a daily clock, reminders armed, workflow notifications on form status change, two cube exports, and a stop/archive pair held disabled.

Read this one when you care about planning cycles, governance and approvals, or getting numbers back into the model.

What contribution campaigns are for

The problem they solve, stated plainly:

> You need to collect data from people who are not necessarily licensed in your system once a quarter or year, and need a platform to take care of that process for you.

The load-bearing phrase is **not necessarily licensed in your system**. The pitch is not the form. It is reaching contributors who have no seat in the planning tool, and running the collection process around them.

Real-world cases this shape covers, drawn from finance teams who have run it:

- **Weekly forecast collection from market presidents.** A short-cycle, recurring collection from senior people who would never log into the planning system.
- **Annual sales planning templates.** Once a year, distributed wide.
- Across both, the dominant purpose was **forecasting**.

That is the frame to put around the Budget Input campaign: it is a *cadence* of collection from unlicensed contributors, not merely a budget form.

Where to look

The question	Where to look
"Can people submit numbers back?"	Budget Input, the round trip
"Who signs off?"	Budget Input: approvals, and the unapproved-data switch on export
"How does it reach our model?"	Budget Input, the two exports
"Will it cope with our headcount?"	Campaigns at large scale, and the profiling and stress-testing screens
"Can the form validate what they type?"	Budget Input, the *"Not a number"* check
"Can they work offline in Excel?"	Budget Input, export/import is on

The round trip, end to end

Worth walking end to end:

1. As an administrator, open **Segment & Channel Budget Input** and look at the three generated forms and who they are assigned to.
2. Switch to the seat of **a contributor on one of the market forms**. They land straight on their input form. Type a value, then type a letter and watch the validation reject it. Submit. (See Signing in, and who to be for how the demo's seats work.)
3. Switch to that form's **approver**. Review, edit if you like (this campaign allows it), approve.
4. Back as the administrator, run the `Plan1` export and watch the numbers land in the `THC_Sales` cube.
5. Open a report that reads that cube and show the new plan numbers.

That last step is the one people remember: **the number they typed shows up in the board pack.**

Related

- Build a campaign · Update an input form · Export campaign data

Check out the AI features

how-to report author

Timberline's AI story is not "we bolted a chatbot on". It is that **a report can carry its own context**: what it is for, what it answers well, what to send elsewhere, and what to be careful about. And that the AI answers from that rather than from a raw grid.

There are four layers, and the demo works best if you show them in this order.

Layer 1: the provider

One connection: **OpenAI Connection**, model **gpt-5.1**, default 2000 tokens, enabled. Worth thirty seconds at the start so the audience knows where the model lives and that it is configurable.

Layer 2: the workspace vocabulary

The workspace defines the lists every report's AI profile picks from: 12 **audiences** (Executive Leadership, FP&A, Finance, Sales Operations, HR, …), 11 **data types** (Financial, Customer, Product, Workforce, Forecast, Budget, Compliance, …) and 10 **scoring categories** (Financial, Customer, Operational, Executive, **Sensitivity**, …).

The point: authors pick from a shared vocabulary, so the corpus stays consistent instead of every report inventing its own labels.

Layer 3: the report's own context

This is the demo. Open **Timberline Sales Information** and show two things side by side.

The AI Context worksheet in the workbook: 31 rows the author maintains where they work, stating:

> **Purpose.** Monthly sales analysis for FY2026 (Actual), with FY2025 for comparison…
>
> **Questions this report answers well.** What drove sales in a given month, and by which product line or channel; month-over-month and year-over-year trend; where growth is concentrated; revenue mix; seasonality.
>
> **Questions to send elsewhere.** Weekly detail or a specific invoice (use the SQL journal); forward-looking Plan or Forecast (this report is Actuals only); profitability below gross margin (use the Income Statement).
>
> **Data caveats.** Net Sales = Gross − Discounts; Margin % is a ratio and must be recomputed on totals, never summed; months are fiscal 4-4-5 periods, not calendar months; totals reconcile across product, channel and market every month.

The AI profile on the job: the same guidance, structured: audiences, data types, relevance scores, *good at*, *avoid*, *warnings*.

Then make the argument: **the "avoid" list is the interesting half.** Telling the model what *not* to answer is what stops a confident wrong answer from a report that does not hold the data. Ask the Assistant something the report cannot answer (a specific invoice, or a forecast number) and show it route you elsewhere instead of guessing.

Other reports carry lighter profiles: the board packs are *good at high-level regional and total company performance*, *avoid detailed root-cause variance*. The scorecard is *good at rep performance and reporting lines*.

Layer 4: AI inside the delivered email

Two jobs generate commentary and put it **in the email body**, not in an attachment.

Sales Rep Scorecard: a prompt called `TopBottom`, handed the team grid, asked for:

> concise commentary outlining our top and bottom 3 sales representatives … written like a management review with fully concise sentences in top and bottom sections, with a one-sentence starting point on overall team performance.

Timberline Sales Information: a prompt called `ProductSales`, handed the product grid **as a Markdown table**, asked for key trends and concerns for the year versus last year, calling out the top positive and top concerning trend.

Run the scorecard, then open the demo inbox. Each manager's mail carries a paragraph about *their own team*, next to their own attainment number. That is the moment that lands: not "AI can summarize things", but **"every manager got personalized commentary on their numbers, automatically, in the email they were already going to receive."**

Two craft details worth pointing at:

- **Both prompts specify format, length and stance.** A management-review voice, a fixed structure, a named number of items. Vague prompts produce vague commentary.
- **One hands the model a raw range, the other a Markdown table.** Markdown is generally better for a wide grid, because the model sees column headers alongside every row.

The suggested run order

1. Provider connection. Thirty seconds.
2. The AI Context sheet and the AI profile on *Timberline Sales Information*. That is the argument.
3. Ask the Assistant a question the report answers well, then one it should refuse.
4. Run *Sales Rep Scorecard* and read a manager's mail in the demo inbox. That is the payoff.
5. If the audience is technical, continue to Use the MCP server.

Before you demo

- The insights are **generated at run time**, so the provider must be reachable and the key valid.
- The source data must be populated, or the model gets an empty grid and says so.
- **Anything in the grid goes to the provider.** The workspace has a *Sensitivity* scoring category for exactly this conversation. Have it deliberately rather than being asked.
- 2000 tokens is a real cap; a very wide grid will truncate.

Related

- Use the MCP server · Build a compelling email dashboard
- Customer guide: AI & Integrations

How Timberline rolls the reporting year forward

explanation · report author · administrator

A use case: the feature is global parameters. This is what Timberline Harvest Co. does with it.

The scenario

Timberline Harvest Co. closes its fiscal year in December, and in the first week of January the finance team has the same chore every time. The board pack, the market statements, the sales analysis and the budget collection all have to move to the new year.

There are only four of them, which is exactly what makes it dangerous. Four is few enough that nobody builds a process, and enough that somebody forgets one. The failure is never dramatic: it is a set of statements that go out labeled with last year, filed into last year's folder, and nobody notices until a regional controller asks why their numbers look familiar.

So the year is not stored on each report. It is stored once, and the reports read it.

How it is wired

One global parameter, `Current Year`, defined as a subset of the time dimension in the planning model rather than as a typed-in value.

Four pieces of content reference it:

Consumer	Uses it as
Financial Statements	its `Year` parameter
Financial Statements, delivered to SharePoint	its `Year` parameter
Timberline Sales Information	its `Year` parameter
Segment & Channel Budget Input (a contribution campaign)	its `Year` parameter

The detail that does the work: because the parameter points at a **subset in the model** rather than at a literal year, it tracks the planning system. Roll the subset forward in the model and every consumer follows on its next run, with nobody opening Reportworq at all.

And because the year is a parameter, it reaches further than the numbers. In these reports the same value drives:

- the **output file name**, `{Year}-{Market} Financial Statements`
- the **email subject** line
- the **destination folder path**, so output files itself into a folder for the new year, created on the way past

That is usually the part people had not thought about. Rolling the year does not just change the figures; it re-labels and re-files the output.

The deliberate contrast

Not everything in Timberline uses it, and that is on purpose.

The **Board Packet** declares its own `Year`, reading the same subset directly. The **Sales Rep Scorecard** goes further and reads the year out of a control cube in the planning model with its own expression.

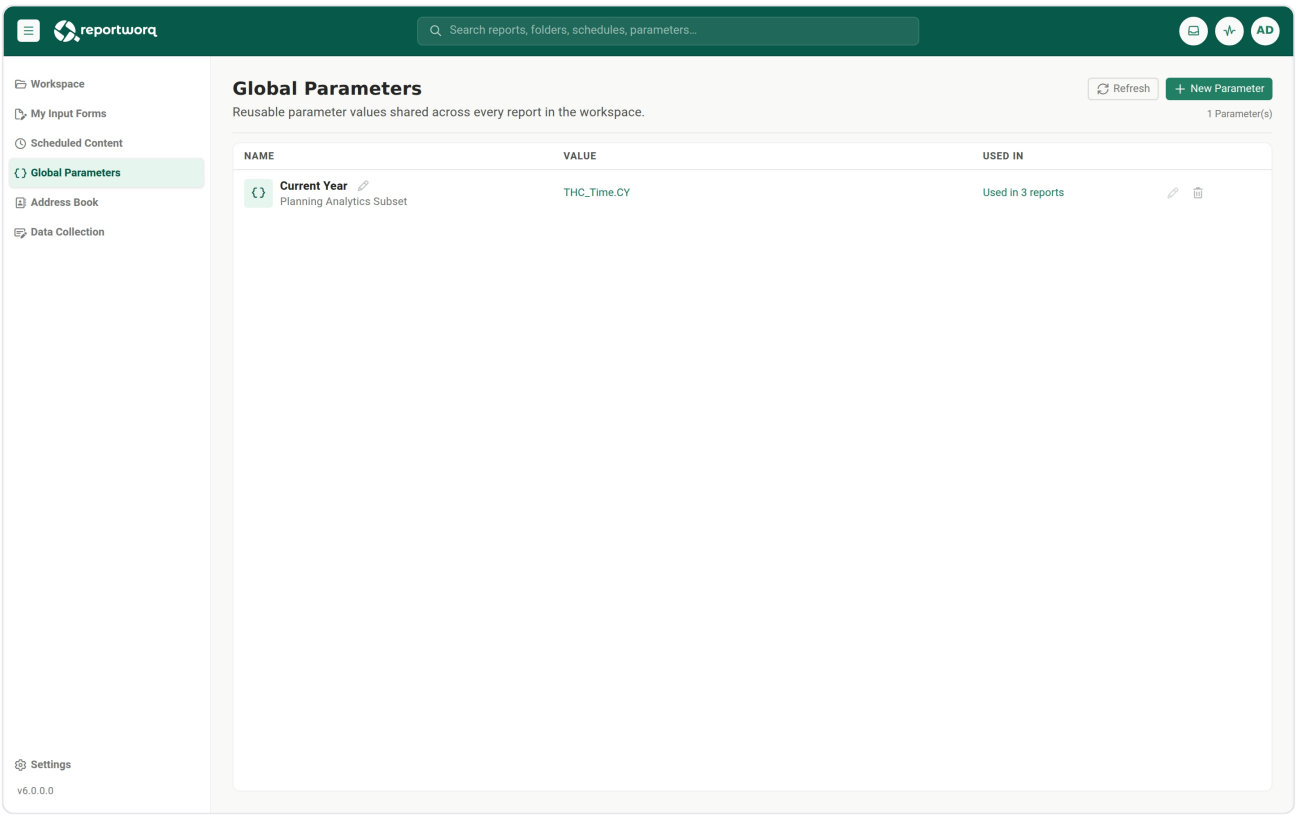
Three approaches, side by side, so you can see the trade-off rather than be told it:

Approach	Use it when
Reference the global	Several reports should always agree, and should move together
Declare it locally	This report legitimately needs its own scope: a prior-year comparison, a fixed historical statement, a period chosen at run time
Read it from the model	The source system already owns the answer and you want one less thing to maintain

The useful question to ask of your own workspace is the one this arrangement encodes: *which of my reports must always be on the same period, and which genuinely need their own?*

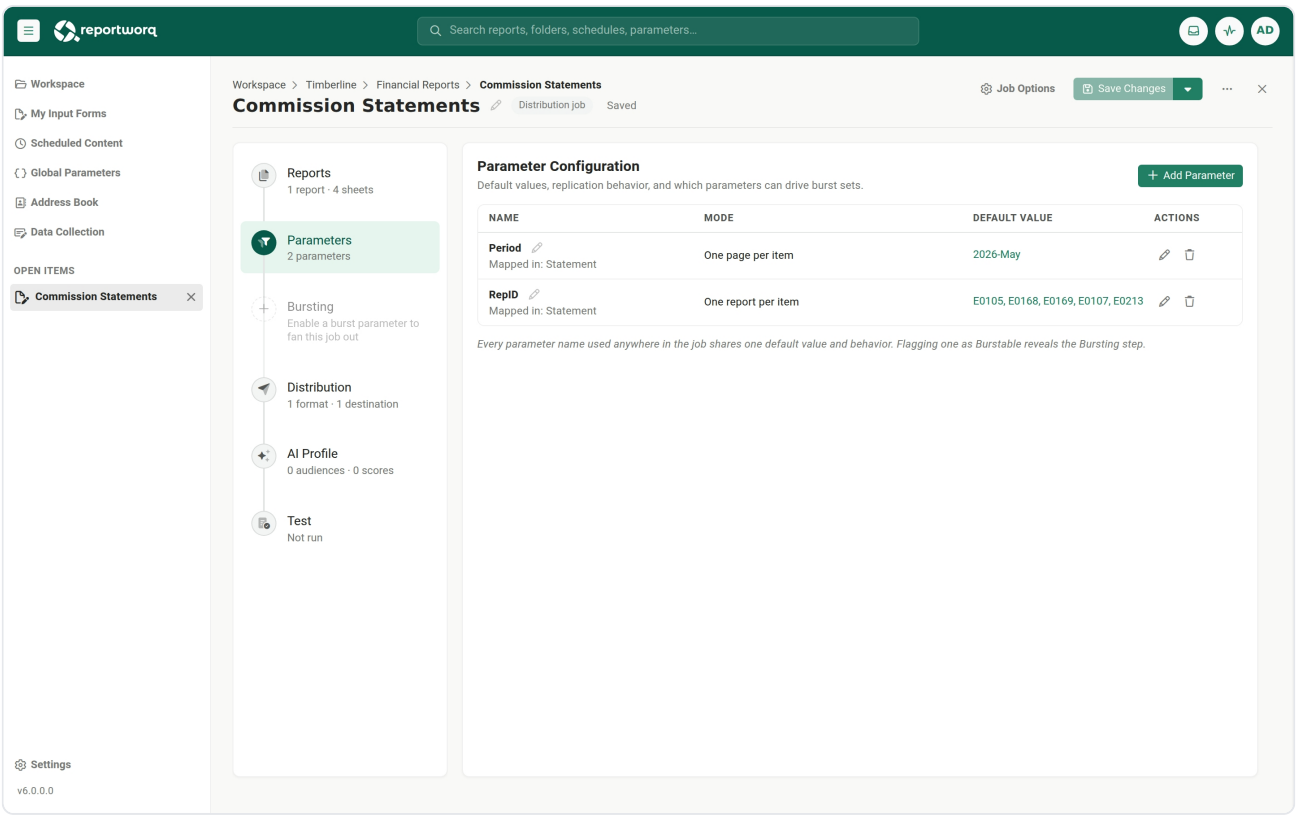
See it yourself

1. Open the workspace and select **Global Parameters** in the left side rail.



2. Find `Current Year`. Look at the **Used In** column: it shows a live count of the reports referencing it. Select the count to list them.

3. Open **Financial Statements** and go to its **Parameters** step. Its `Year` is a *reference* to the global, not its own definition.



4. Now open the **Segment & Channel Budget Input** campaign and look at its parameters too. Campaigns reference global parameters exactly as reports do, which is the point most people miss: the planning cycle moves with the reporting year automatically.

5. Open **Board Packet** and compare. Its `Year` is defined locally. Same value today, different maintenance story tomorrow.

Change it and watch

The demonstration worth doing, because it takes a minute and it lands:

1. On **Global Parameters**, edit the value of `Current Year`.
2. Reportworq shows an **impact review** before it commits, listing every affected report. Read it. This is the safety net that makes a shared value safe to own, and it is the reason you can keep the list short rather than defensive.
3. Confirm, then run **Financial Statements**.
4. Watch three things move together:
    - the numbers, as expected,
    - the **output file name**, which now carries the new year,
    - the **destination folder**, which is created for the new year rather than reusing the old one.
5. Then open the **Segment & Channel Budget Input** campaign and confirm its forms have moved too.

One edit, four consumers, and the file names and folders came along. Set the value back when you are done.

Take it back to your own workspace

The transferable shape is small:

- **Put the value where it is already maintained.** Pointing at a subset or a control value in the source system beats typing a year into Reportworq, because the source system is where somebody already owns the answer.
- **Reference it from everything that must agree**, and let the things that need their own scope keep it. Consistency is not the same as uniformity.
- **Check the impact review before changing a shared value.** In a demo the affected list is four items; in a real workspace it is the difference between a routine edit and an incident.
- **Remember the value reaches the file names and the folders**, not only the figures. That is usually what makes the roll-forward feel finished rather than half done.

Related

- The feature: Global parameters
- In this guide: Use a global parameter · How parameters work

TIMBERLINE DEMO SYSTEM · ORIENTATION

Signing in, and who to be

how-to | report author | administrator

Timberline ships an administrator seat and eight business accounts. Which one you are acting as changes what you can do and what you can see, which is the point, "what does this look like for the person receiving it?" is a question the demo is built to answer.

> **There is only one seat you can actually sign in as.** The administrator account is the one with a password; the eight business accounts have **no password set at all**, so signing in as a contributor or an approver is not possible. To see what they received, read the demo inbox instead, below. Ask your Reportworq contact for the demo administrator credential.

The seats

The demo carries a small identity model, deliberately shaped so that scope is visible: change the seat, change what is in scope.

Seat	Scope	What it is for
Administrator	all workspaces	**Start here.** Authoring and configuration. Every job and campaign in the demo was built from here, and it is the only seat you can sign in as.
Leadership	company-wide	A business seat with company-wide scope. The budget campaign sends its invitations *from* this seat.
Regional and territory seats	one region each	Six of them, each in its own group, so a group maps to exactly one scope. That is what makes secured lists demonstrable.
Contributors	one market each	Three of them, one per generated budget form.
Approvers	one or more markets	Two of them. One approver covers both US markets, which is the more realistic shape: approval rarely maps one-to-one onto entry.

Each group has exactly one member. In a real deployment groups hold many people; here the one-to-one mapping keeps the demo legible.

> **Ask your Reportworq contact for the account list.** Which named account holds which seat is not published here. You do not need it to follow this guide, and the roles above are what the configuration actually keys on.

What each seat can do

Entitlement	Who has it
Administrator	the administrator seat
Workspace Member	all eight business accounts
User	the region and function groups
Author	**nobody**

Two things to know before you demo from a business seat:

- **Nobody holds *Author*.** The business accounts can read, contribute and refresh PowerPoint, but they cannot author. If you want to show authoring from a non-admin seat, grant the entitlement first.
- **Contribution and PowerPoint are not separate entitlements.** Both are capabilities of the **User** role. A User participates in contribution campaigns when the installation licenses the **Contribution** capability, and any User can refresh a distributed PowerPoint deck in the add-in. See Licensing and entitlements.

> **The demo store predates the entitlement consolidation.** It was built when the roles were named System Administrator, Reader, Contribution End User, and PowerPoint End User. Those grants are still in the store, but the product now shows you **Administrator**, **Author**, **User**, and **Workspace Member**. If you are reading the raw store rather than the Security screen, expect the older names. See Roles and what you can do.

Seeing it as somebody else

Just read their mail

Usually the fastest, and it needs no seat switching at all. Because email runs in **demo-inbox mode**, everything the jobs have ever delivered is sitting in an inbox inside the application, addressed to whoever received it, with the embedded screenshots, HTML tables and AI commentary intact.

Use this for: "show me what they actually got".

Everything scopes per user

The point the demo is built to make is that scope follows the person, not the screen. A regional seat sees its own region in a secured list; a contributor sees their own form; AI and integration access answer within the same boundary rather than around it. You do not have to switch seats to believe it, the demo inbox already shows the same job producing different content for different people.

> **Switching seats is an administrator facility.** The business accounts have no password set, so there is no signing in as them. Ask your Reportworq contact to show you the contributor and approver experience directly.

Recipients are not users

This trips people up. The jobs resolve their recipients from the **planning model**, not from the account list, the workbook looks the address up from the employee dimension. So the demo inbox contains mail addressed to people who have no Reportworq account whatsoever.

That is correct, and it is worth saying out loud in a demo: **a report recipient does not need to be a user.** Only people who sign in, contributors, approvers, readers, need accounts. It is usually a licensing question in disguise.

Related

- The campaign catalog, how the forms are assigned
- Build a compelling email dashboard, what is in that mail

How parameters work

explanation report author

A parameter is a value the job pushes into a cell in the workbook before it calculates. Change the cell, the formulas recalculate, and you get a different report from the same template. Everything else about parameters is elaboration on that.

Timberline uses every value source and both replication modes, so it doubles as a reference.

Where a parameter's values come from

Source	Example in Timberline	Reach for it when
Text list	Board Packet's `Region`: `Total Company, North America, EMEA, APAC`	the list is short, stable, and hand-picked
Model subset	Board Packet's `Year`: the `CY` subset of `THC_Time`	the model already maintains the list
Model elements	Budget campaign's `Market`: three named elements of `THC_Market`	you want a hand-picked slice of a dimension
MDX	Sales Rep Scorecard's `Manager`; Divisional Packet's `Region`	the list is a *query*, filtered, drilled, or dependent
Global parameter	`Year` on three jobs and one campaign	many things share one value

The lesson in that table is the second column: **prefer a source the model already owns.** A text list is something you now have to maintain; a subset or an MDX query is not.

What replication does

Every parameter has a replication mode, and it decides the shape of the output:

- **One report per item**: a separate output file, and a separate delivery, per value.
- **One page per item**: the worksheet repeats *inside* one output.

Board Packet sets `Region` to one report per item over four values, and produces four decks. Board Packet with Division Slides stays at total-company level and sets `Market` to one page per item over the same four values, and produces one deck whose sales revenue slide repeats once per division.

A job can use both at once. Divisional Packet gives each region its own file, containing a page per market.

Those last two are worth opening together, because they are the same idea pointed in different directions: the division-slides deck repeats divisions **inside one output**, and the divisional packet repeats **the output itself** per division, then repeats markets inside each one.

The pattern worth learning: a parameter inside a parameter

This is Timberline's best trick. In **Divisional Packet**:

```
Region : {DrillDownMember({[THC_Market].[Total Company]}, {[THC_Market].[Total Company]})}
Market : {DrillDownMember({[THC_Market].[%param:Region%]},  {[THC_Market].[%param:Region
%]})}
```

`Region` drills one level down from Total Company, giving North America, EMEA and APAC: one output each. Then `Market` drills down from **`%param:Region%`**, the value of the output currently being built. So the market list is re-evaluated for every region, and each divisional pack contains exactly its own markets.

Nothing is hard-coded. Add a region to the model and a pack appears for it.

Sales Rep Scorecard does the same over people:

```
Manager : FILTER( leaves of [THC_Employee], [THC_Employee].[Role]       = "Manager" )
Rep     : FILTER( leaves of [THC_Employee], [THC_Employee].[Manager ID] = "%param:Manager
%" )
```

One report per manager, one page per direct report, and the org chart stays true because it *is* the org chart.

A third variant reads a single value out of a control cube rather than expanding a list:

```
Year : {StrToMember("[THC_Time].[THC_Time].[" & [THC_Control].([Controls].[Current Year],
[Control_m].[Value]) & "]")}
```

Roll the year in the model, and every report that reads it follows.

Size before you schedule. Cascading multiplies. Three regions with four markets each is twelve pages; the same pattern over a large dimension fans out fast.

Getting the value into the workbook

A parameter is only a value until a worksheet maps it to a place in the sheet. On the **Reports** step, each worksheet lists its parameters, and the **value field** on a row is where you set that place. Click it and a picker opens, listing everything on that sheet a parameter can point at:

Column	What it offers
Sheet ranges *(preferred)*	named ranges defined on *this* worksheet
Workbook ranges	workbook-level named ranges that land on this worksheet
Pivot fields	a pivot table's page/row/column fields, shown as `PivotTable.Field`

Empty columns are hidden, so a sheet with no pivot simply shows its ranges. You can also just **type** a cell like `F2`, a range, or leave it blank if the value doesn't need to appear in the sheet.

Timberline shows the range styles:

Mapping style	Example
to a **named range**	`Market → Market`, `Rep → RepID`
to a **raw address**	`Region → F2`, `Product → B3`
the same parameter into **two sheets**	`Employee → Employee` and `Employee → Employee2`
the same parameter to **different cells per sheet**	`Market → Market` on three tabs, `Market → B3` on another

There is also a lesser-known option the picker now surfaces: **map a parameter to a pivot field** (`PivotTable.Field`) instead of a cell. The value is applied as a *pivot filter*: a page field jumps to that item; a row or column field filters to it, so a single pivot report re-slices per value without a helper cell.

Two rules follow:

1. **One target per worksheet.** A parameter maps to a single place on each sheet. Needing the same scope on two sheets means two named ranges, which is exactly why the compensation form has `Employee` and `Employee2`.
2. **Prefer named ranges to addresses.** A named range survives someone inserting a row; `F2` does not, which is why the picker marks sheet ranges *preferred*.

> Reorder a worksheet's parameter rows by dragging the grip on the left of a row; remove one with the trash icon on the right.

Parameters do more than fill cells

The same values drive the output around the report:

Where	Example
File names	`{Year}-{Market} Financial Statements`
Worksheet names	`%param:Market% Revenue Review`, `%param:Employee%-Comp`
PDF table-of-contents titles	`%param:Region%-Board Deck`
Destination folder paths	`…/Timberline/{Market}/{Year}`, the folder tree is created from the values
Email subject and body	`{Region}-Board Packet`
Combine keys	Divisional Packet groups emails by `{Region}`

The destination-path one is the underrated feature. One job, and the output arrives filed by market and by year without anyone creating a folder.

Value-level security

Timberline also carries two **secured lists**, which map values to groups so what a user may select depends on who they are:

- **MarketControl**: refreshed from the model by MDX. Leadership sees everything; each region group sees its own region.
- **SalesTerritory**: hand-maintained. Four territories, each mapped to its territory group, its region group and Leadership.

The pair is deliberate: one refreshes itself, one is maintained by hand, and that is the same choice a customer has to make.

> Note: at the time of writing neither list is bound to a parameter on a running job, so they are demonstrable on the administration screen but you cannot yet show one filtering a report.

Related

- What bursting looks like · Use a global parameter
- Customer guide: Advanced parameters

Build a campaign

how-to report author

Building a contribution campaign is seven decisions. Timberline's **Segment & Channel Budget Input** is a worked answer to all seven, so the fastest way to learn is to build alongside it, open it in one tab and follow this in another.

1. Start from a form template, not a report

A campaign's source is the workbook contributors will type into. It is a normal Excel workbook with two extra things:

- **Input bindings**: a formula that says "this range of cells writes to this model, along these dimensions". In the budget form:

  ```
  =RW_CONTRIBUTION_INPUT($D$10:$D$33, "volume",
       "Account", $A:$A, "Channel", $B:$B, "Market", $B$1, "Time", $B$2, "Measure",
  "Volume")
  ```

  Read it as: *range*, then *model name*, then alternating *dimension, source*. A source can be a **column** (the value varies down the range) or a **cell or literal** (fixed for the whole block).

- **Validation**: optional per-cell checks:

  ```
  =rw.contribution.check(D10, ISNUMBER(D10), "Not a number", FALSE)
  ```

  Cell, test, message, and whether failing blocks submission.

Keep these formulas in hidden rows or columns. Contributors should see the grid, not the plumbing.

2. Decide what one form covers

This is the parameter decision, and it determines how many forms you generate and who owns each one.

The budget campaign sets `Market` to **one report per item** over three chosen model elements, so **one form per market**. Had it been set to an employee dimension instead, you would get **one form per person**.

Ask: *what is the smallest unit somebody owns end to end?* That is your form.

The budget campaign also sets `Year` to **one page per item** from the **Current Year** global parameter, so the year is scope rather than a separate form, and rolling the year re-points the whole campaign from one place.

3. Name the forms so people recognize them

The form file name is built from the parameters, `{Market} Segment and Channel Input`, producing *USA - East Segment and Channel Input.xlsx*. A contributor with several forms should be able to tell them apart at a glance.

Worksheet names can be parameterized too: the compensation form renames its sheets `%param:Employee%-Comp` and `%param:Employee%-Sales`.

4. Decide whether anyone approves

The budget campaign requires approvals **and** allows approvers to edit forms. That combination says: the approver is a reviewer with authority to correct, not a rubber stamp. The compensation campaign has no approvals at all.

Approval is not only a workflow decision, it changes what your export can trust. See step 6.

5. Generate and assign

Generating creates one form per parameter value. Assigning attaches a **contributor** and, if you required approvals, an **approver** to each.

Timberline assigns individuals, one per form. You can also assign groups, which is what you want when several people share ownership.

The shape worth copying: contributors are the regional people, approvers sit one level up and cover more than one form. In the demo, one approver covers both US markets.

6. Wire the write-back

An **export action** maps collected data into a cube. The budget campaign has two, and they differ on purpose:

	`Plan1`	`FixedCosts`
Target cube	`THC_Sales`	`THC_IncomeStatement`
Trigger	**none**: run on demand	**form status change**
Includes unapproved data	**no**	**yes**
Write-back	proportional	proportional

`Plan1` is the *end-of-cycle* export: run it once the numbers are approved, and it takes only approved data. `FixedCosts` is the *continuous* export: it fires whenever a form moves state, keeping the model warm while people are still working, and accepts that some of what it writes is not yet approved.

Choosing between those two shapes is the real design decision. See Export campaign data for the dimension mapping.

7. Automate the human parts

The budget campaign carries ten tasks:

Task	Trigger	Purpose
contributor / approver **invitation**	daily 08:00	tell people they have a form
contributor / approver **reminder**	armed, no clock	chase-ups when you want them
contributor / approver **workflow**	form status change	notify when a form moves
`Plan1`, `FixedCosts`	on demand / status change	the exports
stop campaign, **auto archive**	disabled	the wind-down pair

All six notifications use the shared **message templates**, which reference campaign fields: project name, form name, and a form link that takes the recipient straight to their form. One template serves every campaign.

Configure the wind-down pair even if you leave it disabled. Knowing how the cycle *ends* (stop input, then archive) is part of designing it.

Before you call it done

- Generate, then open a form **as the contributor** and check it reads as a form, not as a spreadsheet with plumbing showing.

- Type something invalid and confirm the validation fires.
- Submit, approve, export, and check the number in the cube.
- If the campaign is large, size it before you invite anyone: see Campaigns at large scale.

Related

- Update an input form · Export campaign data
- Customer guide: Author a contribution campaign

Use the MCP server

how-to | report author | administrator

MCP lets an AI client (Claude, Microsoft 365 Copilot, or anything else that speaks the protocol) call Reportworq directly: list content, run a job, fetch output, ask about a report. Timberline is almost entirely pre-configured for it, with one deliberate exception.

The one switch

`AllowMcpAccess` **is off in the shipped demo store.** Nothing MCP works until an administrator turns it on in the authentication settings.

That is the safe default for a shared workspace, but it means **MCP needs one deliberate setup step before you can use it**.

What is already configured

Setting	State	What it means
CloudHub channel	enabled	the hosted route is available
Direct channel	enabled	the direct route is available
Test-user impersonation	**enabled**	an administrator can exercise MCP as another user
Public download links	allowed	generated output links do not require sign-in
Download token lifetime	900 seconds	links expire after 15 minutes
Download token use	single-use	a link works once
Inline output cap	200,000 bytes	larger output becomes a link instead of inline content
MCP audit logging	on, 90-day retention	every call is recorded

So once access is on, both channels, impersonation, auditing and sane link limits are already in place.

Note the two demo conveniences that are **not** recommendations: public download links and a never-expiring authenticated-link setting. Say so if the room is security-minded. The point is that these are per-deployment choices, and yours would be different.

Setting up a demo

1. Sign in as an administrator and **enable MCP access** in the authentication settings.
2. Pick a channel. **Direct** is the simpler story on a laptop; **CloudHub** is the one to show if the conversation is about a hosted deployment.
3. Connect your client. The mechanics differ per client. See the customer AI & Integrations guide for the client-side setup, which this guide does not duplicate.
4. Verify with something read-only first: list the workspace content, and confirm the eight jobs and two campaigns come back.
5. Check the **MCP audit** afterwards and show the calls that were made. It is a good closing beat for a governance-minded audience.

What to actually ask it

Point questions at content that was *built* to answer them.

The answer-layer report templates that ship with the demo exist largely for this. Six parameterized workbooks (sales trend, income statement, margin mix, balance sheet and cash flow, and sales ops), each with:

- an **AI Context** tab describing what it is and is not for,
- a **Control** tab with named parameter cells (`ReportYear`, `MarketSel`, `VersionSel`),
- an embedded data tab tagged with a geography key, so one formula scopes to Total Company, a region or a market,
- report tabs whose columns run `Jan Feb Mar Q1 … Q4, FY, prior year, YoY%`.

They are granular and self-describing, which is exactly what an MCP question needs. Ratios are always recomputed rather than summed, so a margin question gets a correct answer rather than an averaged one.

The jobs themselves carry AI profiles, so questions about *which report should I use* have something to work with. See Check out the AI features.

Good demo questions:

- *"What drove sales in February, and by which product line?"* Sales Information answers this well.
- *"What was FY2026 gross margin for EMEA?"* A scoped answer from the answer layer.
- *"Run the Financial Statements job for North America and give me the PDF."* The run-and-fetch path, which is the one people actually want to see.
- *"Which report should I use for a specific invoice?"* This should route you away, because the profiles say so.

Acting as another user

With impersonation enabled, an administrator can exercise MCP **as** a specific user without collecting passwords. Use it to show that AI access is scoped: ask the same question as a Leadership user and as an APAC user, and show the difference.

This is a testing and demonstration facility. Treat it as an administrator capability, and mention the audit log in the same breath: impersonated calls are recorded.

Before the meeting

- **Turn MCP access on**, and check it is on. This is the step people forget.
- Confirm the AI provider connection is valid: MCP and the in-app AI both depend on it.
- Confirm the job's data source is reachable if you plan to run a job rather than just list content.
- Decide whether you are demoing the direct or CloudHub channel, and set the client up in advance.

Related

- Check out the AI features
- Customer guide: AI & Integrations: Copilot & MCP

TIMBERLINE DEMO SYSTEM · ORIENTATION

Where the files live

how-to | report author | administrator

Every Timberline report is a workbook, and every workbook is stored in one of two places. Knowing which is the difference between "I edited it and everything still works" and "the demo failed in front of a customer".

The nine source files

They are in **Workspace ▸ Files ▸ Timberline ▸ Financial Reports ▸ Source Files**:

File	What it is	Used by
`Timberline_Board_Pack.xlsx`	the board pack, 4 chart sheets plus a hidden data sheet	the three board-deck jobs
`Timberline_Board_Pack_Market.xlsx`	the per-market revenue review	the two division-slide jobs
`Timberline_Board_Pack.pptx`	the board deck template	Board Packet, Board Packet with Division Slides
`TimberlineDivisional_Board_Pack.pptx`	a variant deck template	*(nothing currently, see the Codex note)*
`Financial_Board_Book.xlsx`	income statement, balance sheet, cash flow	both Financial Statements jobs
`Timberline Sales Information.xlsx`	the deep sales analysis, plus an AI Context sheet	Timberline Sales Information
`Sales Rep Scorecard.xlsx`	team and rep scorecards	Sales Rep Scorecard
`Timberline_Budget_Input_Model.xlsx`	the budget contribution form template	Segment & Channel Budget Input

The two storage locations, and why it matters

Workspace file storage holds seven of the nine. These live *inside* the workspace, so they travel with it: stand the workspace up anywhere and the files are there.

Network Folders holds the other sources. A provider called *Reports* points at a folder on the server's filesystem. Two things read from it: the **Commission Statements** workbook, and the PowerPoint template used by **Divisional Packet**.

The consequence is the general lesson: **workspace file storage travels with the workspace; a network folder does not.** A job bound to a network provider needs that folder to exist, with the service account able to read it. If you move a workspace between servers, the workspace files come along and the network-folder sources have to be re-created.

How to change a workbook

Do it through the application, not by editing files on disk. Editing the stored bytes directly desynchronises the file record and bypasses the store's change tracking.

1. **Download** the workbook from **Workspace ▸ Files ▸ … ▸ Source Files**.
2. **Edit it in Excel**, with the Reportworq add-in installed. The workbooks are full of add-in formulas, and saving them without the add-in available bakes errors into the cells.
3. **Upload it back over the same file record**, so the file's identity is preserved. Every job binds to a file **id**, not to a filename; replacing the record keeps the binding, adding a new file breaks it.
4. **Re-run Test / preview** on every job that uses the file. The list is in the table above.

For the Network Folder sources, replace the file at the network root on the server instead.

The one rule: do not break the names

Jobs bind to named ranges and worksheet names, not to cell addresses. Rename a sheet, delete a named range, or move a range a job maps a parameter to, and the job breaks *at run time*, not when you save. Nothing warns you.

These are the names currently load-bearing:

Workbook	Names a job depends on
`Timberline_Board_Pack.xlsx`	`EmailTO`, `FirstName`, `SlideTitle`, `StartingYear`, `SummaryPL`, `NetRevKPI`, `GMKPI`, `OMKPI`, `NIKPI`, `NMKPI`, `CAGRKPI`, `IncomeStmt`, `Outlook1`, `OUtlook2`, `PlanAssumptions`
`Timberline_Board_Pack_Market.xlsx`	`Market`, `IncomeStmt`, `ProductSlice`
`Financial_Board_Book.xlsx`	`Market`, `Year`, `EmailTo`
`Timberline Sales Information.xlsx`	`Market`, `Channel`, `Product`, `Year`, `EmailTO`, `EmpName`
`Sales Rep Scorecard.xlsx`	`Manager`, `RepID`, `CY`, `MgrName`, `MgrEmail`, plus the report-view names below
`LargeContributionForm.xlsx`	`Employee`, `Employee2`, `Suppress`, plus the report-view names below

Both of those workbooks also carry **reserved names that the add-in generates for you**, controlling how a dynamic-row grid expands. You will recognize them in the Name Manager because they are prefixed and look nothing like the readable names above. **Leave them alone**: they are not yours to rename, and a report that loses one stops expanding its rows.

Two traps worth knowing before you edit

Sheet-scoped names shadow workbook-scoped ones. `Timberline Sales Information.xlsx` defines `Market`, `Year`, `Product` and `Channel` at workbook level *and* redefines several of them at sheet level on three of its tabs. That is how one parameter drives a different cell on each tab, but it means "find the Market cell" has more than one right answer, and changing the wrong one silently affects only some tabs.

A parameter maps to one cell per worksheet. That is why `LargeContributionForm.xlsx` has both `Employee` (on the Comp sheet) and `Employee2` (on the Sales sheet) for what is conceptually one value. Any template that needs the same scope on two sheets needs two names.

Hidden and excluded sheets

Two sheets do unusual jobs, and neither is a mistake:

- **The board pack's data sheet** is **hidden but included**. It holds the parameter cells, the recipient lookups, the slide title and the data reads that the visible chart sheets reference. Excluding it breaks the deck; hiding it keeps it out of the output.

- `AI Context` in the sales information workbook is **visible but excluded**. It carries 31 rows of guidance for the AI, purpose, questions it answers well, questions to send elsewhere, data caveats, so the context lives with the report and travels with it. It never appears in output.

Related

- Start here · The report catalog

What bursting looks like

explanation report author

Bursting is one job producing many personalized outputs and delivering each to the right person. Timberline does it eight times over, and it is worth being precise about *how*, because the mechanism is the thing customers get wrong.

The headline: Timberline has no burst lists

Open any Timberline job and look at the bursting screen. There is nothing in it.

Every job here fans out from its parameters, driven by the planning model, not from a burst set someone typed in. That is deliberate, and it is the single most useful thing to show, because the assumption most people arrive with is that bursting means maintaining a list of who gets what.

- *Commission Statements* produces a statement per rep because `RepID` has values.
- *Sales Rep Scorecard* produces a scorecard per manager because an MDX filter says who the managers are.
- *Divisional Packet* produces a pack per region because an MDX drill-down says what the regions are.

Nobody maintains any of those lists. They are the model's lists.

Burst sets still have their place (a curated external roster with no home in a source system), but Timberline deliberately does not use one, so the demo does not teach the wrong habit.

Where the recipient comes from

The second half of bursting is routing: which output goes to whom. Timberline answers that from the **workbook**, not from a contact list.

Each source workbook contains a named range whose formula chains through the planning model's element attributes: it reads the **VP code** attribute of the current `Market`, then reads the **Email** attribute of that person, one lookup nested inside the other.

Read outward: take the market, look up its VP code, look up that person's email. The job's **Email to** field points at that named range, so each burst addresses itself.

Job	Recipient comes from
Board Packet, Divisional Packet	`EmailTO` on the hidden data sheet
Financial Statements	`EmailTo` on the Income Statement sheet
Sales Rep Scorecard	`MgrEmail` on the Team Scorecard
Commission Statements	`ManagerEmail` on the Statement sheet
Timberline Sales Information	`EmailTO` on the Sales by Product sheet

The consequence is worth saying in a demo: **the distribution list is maintained in the planning system, by the people who already maintain the org chart.** When someone changes role, the next run is already correct.

Watching a burst happen

The cleanest demonstration in the workspace is **Commission Statements**, because you can see the fan-out and the roll-up in the same run.

1. Open the job and look at the parameters: `RepID` has nine values, so nine statements.
2. Run **Test / preview**, you see the nine outputs before anything is delivered.
3. Run the job.
4. Open the **demo inbox**. There are not nine emails. There are a handful, **one per manager**, each containing that manager's whole team's statements.

That last step is the combine key doing its work: the job groups the burst outputs by a `Territory` cell value before sending.

A burst that is not a burst

Board Packet with Division Slides is the useful counter-example. It also produces four versions of the market review, but they all land in *one* deck, as four repeated slide sections, delivered once.

That is the distinction between the two replication modes:

- **one report per item** → many files, many deliveries, a burst
- **one page per item** → one file, one delivery, repeated content inside it

People conflate the two constantly. Running Board Packet and Board Packet with Division Slides back to back settles it in about ninety seconds.

Bursting at scale

Commission Statements is configured with nine reps, which is a demo-sized number. The underlying design supports the full 237-person sales organization, the parameter would simply return more values, and the combine key would still roll them up per manager.

Two things to size before scaling a burst like this for real:

- **Output count multiplies with cascading parameters.** One report per manager times one page per rep is fine; one report per rep times one page per period times one page per product is not.
- **Combine changes the delivery count, not the generation count.** The job still generates every output; combine only affects how many messages leave.

Related

- How parameters work
- Customer guide: What bursting is and when you need it

Update an input form

how-to report author

An input form is generated from a template workbook. Changing what contributors see means changing the template and then regenerating, with a decision to make about work already in progress.

The two templates

Template	Campaign	Sheets
`Timberline_Budget_Input_Model.xlsx`	Segment & Channel Budget Input	Budget Entry, Fixed Costs, Summary

Both are in **Workspace ▸ Files ▸ Timberline ▸ Financial Reports ▸ Source Files**.

Change the template

1. **Download** the template from workspace files.
2. **Edit in Excel** with the Reportworq add-in available. The forms are full of add-in formulas; saving them without the add-in bakes errors into the cells.
3. **Upload back over the same file record** so the campaign's binding survives, it binds to the file id, not the name.
4. **Regenerate the forms** in the campaign, then open one as a contributor and check it.

What you are usually changing

Adding or moving input cells

Input cells are defined by a **binding formula** that names the range:

```
=RW_CONTRIBUTION_INPUT($D$10:$D$33, "volume",
     "Account", $A:$A, "Channel", $B:$B, "Market", $B$1, "Time", $B$2, "Measure", "Volume")
```

If you add rows to the grid, **extend the range in the binding to match.** A cell inside the visual grid but outside the bound range looks editable and writes nowhere, the most common way a form goes quietly wrong.

The dimension arguments alternate *dimension, source*. A **column reference** (`$A:$A`) means the value varies down the range; a **cell or literal** (`$B$1`, `"Volume"`) means it is fixed for the whole block. Adding a dimension to the target cube means adding a pair

here.

The budget form has nine bindings, one for the volume/price/cost drivers, five for row-blocks of the plan, one for fixed costs. The compensation form has seven, one per measure.

Adding validation

```
=rw.contribution.check(D10, ISNUMBER(D10), "Not a number", FALSE)
```

Cell to check, the test, the message the contributor sees, and whether a failure blocks submission. The budget form has twenty of these, all numeric checks, all non-blocking.

Put them in a column outside the print area, the budget form keeps them in column L. Contributors should see the message, not the formula.

Changing what a sheet is called, or whether it appears

- **Worksheet names** can be parameterized: `%param:Employee%-Comp`.
- **Suppression** can drop a sheet entirely. The compensation form's Sales sheet is suppressed from a `Suppress` cell, so it disappears for employees it does not apply to.

Adding a calculated block

Not everything on a form is input. The budget form's **Summary** sheet is a read-only contribution-margin waterfall that recalculates as the contributor types, it makes the form feel like a model rather than a data-entry screen, and it is worth copying.

The rules that will bite you

Keep write-back formulas out of sight. Contributors see `#NAME` where a Reportworq function sits, so put bindings and checks in hidden rows or columns outside the print area.

Extend the binding when you extend the grid. See above. This is the big one.

Do not touch the reserved names the add-in generates. The form carries a small family of prefixed names, obvious in the Name Manager because they look nothing like the readable ones, that belong to the report view and control how a dynamic-row grid expands. Renaming or moving them breaks the grid.

One parameter maps to one cell per worksheet. The compensation form needs both `Employee` (Comp) and `Employee2` (Sales) for what is conceptually one value. Adding a third sheet that needs the same scope means a third named range.

Set the print area deliberately. It defines what a contributor sees and what an exported workbook contains. The budget campaign has *hide non-print area* and *lock non-input cells* switched on for export, and both depend on the print area being right.

What happens to forms already in progress

Regenerating replaces the form. Work already entered is held in the campaign's runtime state rather than in the template, but a structural change (moved rows, a renamed sheet, a changed binding range) can mean previously entered values no longer line up.

Safe sequence when a campaign is live:

1. Make the change on a **copy** of the campaign first and generate one form to check it.
2. If the change is structural, tell contributors before regenerating.
3. If the campaign is essentially finished, run the export first so the numbers are safely in the model.

The `auto_archive` action includes a **refresh forms** option, which is the built-in way to regenerate at the end of a cycle. Leave it disabled on the demo store, enabling it archives the campaign.

Related

- Build a campaign · Campaigns at large scale
- Where the files live
- Customer guide: Contribution functions

Use a global parameter

how-to | report author | administrator

A global parameter is a value defined once for the workspace that any job can reference instead of declaring its own. Timberline has exactly one, and it exists to make a specific demo moment possible: **roll the reporting year forward in one place, and watch everything follow.**

The one that exists

Current Year: defined as the `CY` subset of the `THC_Time` dimension on `Planning Sample Demo`.

Four pieces of content reference it:

Consumer	Uses it as
Financial Statements	its `Year` parameter
Financial Statements - SharePoint	its `Year` parameter
Timberline Sales Information	its `Year` parameter
Segment & Channel Budget Input (campaign)	its `Year` parameter

Because the definition points at a **subset**, not at a literal year, the value tracks the planning model. Roll `CY` forward in the model and every consumer follows on the next run without anyone touching Reportworq.

How to demonstrate it

1. As an administrator, open the workspace **global parameters** screen.
2. Show **Current Year** and what it points at, a subset in the model, not a typed value.
3. Open **Financial Statements**, look at its `Year` parameter, and show that it is a *reference* to the global rather than its own definition. Do the same in the **Segment & Channel Budget Input** campaign, so the audience sees that campaigns reference it too.
4. Change the global, or roll `CY` in the model, and re-run. Both the report and the campaign pick up the new year.

The point to land: **three reports and a whole planning campaign re-point from one change.** Without this, that is four edits, and one of them gets forgotten.

The deliberate contrast

Not everything uses it. **Board Packet** declares its own `Year`, reading the same `CY` subset directly. **Sales Rep Scorecard** reads the year out of the `THC_Control` cube with its own MDX expression.

That is not inconsistency, it is the demo showing both options so you can talk about the trade-off:

- **Reference the global** when several pieces of content should always agree, and should move together.
- **Declare it locally** when a report legitimately needs its own scope, a prior-year comparison pack, a fixed historical statement, a report whose year is chosen at run time.

A useful question for the room: *"which of your reports should always be on the same period, and which genuinely need their own?"* That is the decision the global parameter exists to express.

Where else the year shows up

Once a job has a `Year` parameter, the value is available everywhere parameters are:

- in output file names: `{Year}-{Market} Financial Statements`
- in email subjects: `{Year}-{Market} Financial Statements`
- in **destination folder paths**: the SharePoint variant files output into `…/Timberline/{Market}/{Year}`, creating the year folder as it goes
- in the workbook itself, wherever the `Year` named range is referenced

So rolling the year does not just change the numbers. It changes the file names, the subject lines and the folder the output lands in, which is usually what people actually wanted.

A caution

Changing a global parameter affects **every** consumer at once. That is the feature, and it is also the risk: check the consumer list before changing it on a shared environment. In Timberline the list is short and above; in a customer's workspace it may not be.

Related

- How parameters work
- Customer guide: Global parameters

Export campaign data

how-to report author administrator

Collecting numbers is only half a campaign. The export is what puts them back into the planning model, and Timberline demonstrates the two shapes that decision comes in.

The two exports in the budget campaign

	`Plan1`	`FixedCosts`
Target	`Planning Sample Demo` / `THC_Sales`	`Planning Sample Demo` / `THC_IncomeStatement`
Trigger	**none**: run on demand	**form status change**
Includes unapproved data	**no**	**yes**
Includes unchanged data	no	no
Write-back	proportional	proportional

`Plan1` **is the end-of-cycle export.** No trigger, so somebody runs it deliberately once the cycle closes. It takes only approved data, so what lands in the sales cube is what the approvers signed off.

`FixedCosts` **is the continuous export.** It fires whenever a form changes state, and it *includes unapproved data*, so the income-statement cube stays warm while contributors are still working, and downstream reports show current thinking rather than last month's.

That is the design decision: **do you want the model to reflect what is agreed, or what is current?** Most campaigns want one of each, a live view for the people running the process, and a governed view for the numbers of record.

How the mapping works

An export maps each form's collected data into a cube by matching dimensions:

```
Cube:       THC_Sales
Dimensions: THC_Version | THC_Market | THC_Product | THC_Channel | THC_Time |
THC_Sales_Measure

THC_Version        = "Plan"            ← fixed
THC_Market         = Fields.Market     ← from the form
THC_Product        = Fields.Account    ← from the form
THC_Channel        = Fields.Channel    ← from the form
THC_Time           = Fields.Time       ← from the form
THC_Sales_Measure  = "Net Sales"       ← fixed
```

Every dimension of the target cube gets a source, and there are two kinds:

- **A fixed value**: `"Plan"`, `"Net Sales"`. This export always writes plan net sales; nothing about the form can change that.
- **A form field**: `Fields.Market`, `Fields.Account`. These come from the input binding's dimension arguments in the template. `Fields.Account` exists because the binding declared `"Account", $A:$A`.

The names have to line up. The `Fields.*` names are exactly the dimension names used in the template's `RW_CONTRIBUTION_INPUT` bindings. Change a binding's dimension name and the export mapping must change with it, or the export silently loses that dimension.

The `FixedCosts` export is the same pattern against a four-dimension cube: `THC_Version` fixed to Plan, and Market, Time and Account from form fields.

Proportional write-back

Both exports have **proportional write-back** on, and the shared model connection carries it too.

That means a value written at a consolidated level **spreads across its children in proportion to their existing values**. Type a total for a product line and the products underneath it move together, keeping their relative shape.

This is usually what a planner wants, and it reliably surprises people the first time they see it. Say it out loud in a demo rather than letting someone discover it.

Running an export

1. Open the campaign and go to its automation tasks.
2. For an on-demand export like `Plan1`, run it from there once the forms are approved.

3. Check the result in the model, or, better, open a report that reads the cube and show the numbers there. *Timberline Sales Information* reads `THC_Sales`, so contributed plan numbers show up in a report the audience has already seen.

Step 3 is the moment worth building the demo around: **the number they typed appears in the board pack.**

Getting data out as a file instead

The budget campaign has **export/import enabled**, which is a different thing: contributors can take their form out to Excel, work offline, and bring it back. Two options shape what they get:

- **hide non-print area**: the exported workbook shows the form, not the plumbing.
- **lock non-input cells**: they can only type where they are supposed to.

Both are on for the budget campaign. If you turn on protection, keep *restrict to input cells* on too, or the exported workbook locks everything including the cells people need.

Two cautions

- **The export needs write-back rights.** The model connection is shared with reporting; reading works long before writing does. Test the write path separately.
- **Including unapproved data is a governance choice, not a convenience.** Anything the continuous export writes is visible to everything downstream. Be sure the audience for that cube understands they are seeing work in progress.

Related

- Build a campaign · Update an input form
- Customer guide: Contribution overview

Build a compelling email dashboard

how-to report author

Five Timberline jobs build the same shape of email, and it is the most reusable thing in the whole demo. The idea: **the message body is the report; the attachment is for whoever wants to go deeper.**

Three principles first

There is no single right layout; it varies with the report and the audience. Three principles matter more than the template:

- **Put screenshots of the relevant information in the body** so a reader can glance and get an overall understanding of the data.
- **Use AI insights as a summarized starting point** for people to think about their data. A starting point for thought, not a conclusion.
- **Do not make it too wordy**, or it gets skipped, but do include what helps at a glance so the reader knows what the email contains.

The recipe below is a worked instance of those three, not a fixed template.

The recipe

Ingredient	How it is expressed	Timberline example
Recipient, from the data	**Email to** bound to a named range whose formula looks the address up from the model	`EmailTO`
Personal greeting	another named range in the body	`Hi {FirstName},`
One headline number, in a sentence	a cell reference inside prose	`your total team attainment is {G11}`
A picture of the report	a **screenshot** of a cell range, scaled	`{Screenshot: A5:N14 @50% JPEG}`
A real table	a named range rendered as an **HTML table**	`{SummaryPL as HTML table}`
Generated commentary	an **AI insight** prompt	`{AI insight: TopBottom}`
A subject that says which one this is	parameter tokens	`{Region}-Board Packet`
The detail, attached	Excel / PDF / PowerPoint	all formats

The two worked examples

Board Packet: the visual version:

> Hi *Sarah*, Please see attached the Executive Board Packet for your review. Below is a summary of the key KPIs and Overall P&L. **[screenshot of the KPI band, A5:N14 at 50%] [the SummaryPL range as an HTML table]** Thanks! Finance Team

Sales Rep Scorecard: the AI version:

> Hello *Thomas King*, For the *Region: APAC* your total team attainment is *90.8%*. **[generated paragraph naming the top and bottom three reps, in a management-review voice]** Attached is a detailed review. Thanks!

Both are personalized per burst. Every manager's mail carries their own attainment number and a paragraph about their own team.

How to build one

1. **Give the workbook the pieces.** Create named ranges for the things the email needs: the recipient address, the greeting name, any headline cell, and the range you want to show as a table. Put the lookups on a hidden sheet if they are plumbing rather than content, the board pack keeps all of this on its hidden data sheet.
2. **Point *Email to* at the address range** rather than typing an address. This is what makes the burst self-addressing.
3. **Write the body**, inserting variables where you want values. Three kinds matter:
   - a **cell value** for prose and headline numbers,
   - a **cell value as an HTML table** for a small tabular block,
   - a **screenshot of a range** for anything with formatting or conditional color you want preserved.
4. **Parameterise the subject** so the recipient can tell which slice this is: `{Region}-Board Packet` beats "Board Packet".
5. **Add an AI insight** if the report has something to say, see Check out the AI features.
6. **Test / preview**, then run and read the result in the demo inbox.

Choosing between a screenshot and an HTML table

Both appear in the same board-pack email, which is not an accident:

Use a screenshot when	Use an HTML table when
the block has conditional formatting, color, sparklines or a chart	it is plain numbers in rows and columns
the layout matters more than selectable text	the recipient might want to copy a value
the block is wide	the block is narrow enough to reflow

A screenshot is an image: it always looks exactly right and never reflows. An HTML table reflows and stays selectable but loses complex formatting. Board Packet uses a screenshot for the KPI band (which is styled) and a table for the P&L summary (which is not).

The board pack renders its screenshot at **50% scale as JPEG**, tuned for email clients rather than for print. That is a reasonable starting point.

What makes these good, and what to avoid

Good:

- The first line names the recipient and the second gives them a number. Nobody has to open anything to learn whether they need to care.
- The subject line is parameterized, so a manager with three regions can tell the mails apart.
- The commentary is *about their data*, generated per burst: not a static paragraph.

Avoid:

- A wall of attachments and no body. That is the pattern this replaces.
- Very wide ranges as HTML tables: they reflow badly. Screenshot those instead.
- Putting everything in. Two visual blocks is plenty; the attachment carries the rest.

Related

- What bursting looks like
- Check out the AI features

Campaigns at large scale

how-to | report author | administrator

Three forms is a demo. A real compensation or budget cycle is one form per person, opened by many of them at once.

What "large" actually means

Scale in contribution has two axes, and they cost differently.

Form size: how much is in one form. Every cell a form holds costs memory for as long as somebody has it open. For a sense of what heavy looks like: two sheets of roughly 60 rows with monthly columns runs to about 840 cube reads plus the level formulas that drive the outline.

Concurrency: how many forms are open at the same time. This is the number that decides what the server needs. A thousand forms that are never open at once cost nothing; fifty open simultaneously do.

The rough shape: **cost ≈ bytes per cell × cells per form**, and the number that matters for sizing is the **marginal cost of one more form open at the same time.**

Size it before you invite anyone

There is a loop for this, and skipping it is how a campaign falls over on day one:

1. **Design** the form. Fewer cells beats more; a form nobody can fill in twenty minutes is too big anyway.
2. **Measure** one form's real server-side footprint with the campaign's **memory profiling** view. It is read-only and isolated, so you can run it against the real template.
3. **Tune**: trim rows, columns and sheets that are not load-bearing. Suppression helps: a sheet that drops out for the contributors it does not apply to costs those contributors nothing.
4. **Stress test** with the campaign's stress-testing tools: build a plan, run it against a **disposable copy of the campaign** with a **dedicated test user**, and reset between runs.
5. **Size** from the results. Plan to about 80% of capacity, and remember the worker baseline when sessions map one-to-one to workers.
6. **Watch** it in production.

The analysis view turns node results into plain language, what a typical user experienced, what the slowest user experienced, and where the slow tail is. That last one is what people complain about; averages hide it.

> **Do this on a copy.** Stress testing generates forms and sessions. Never point it at a campaign people are using, and never at the demo store you are about to present from.

The two ceilings

Two limits bite before anything else:

- **A memory admission valve** kicks in around 90% utilization and starts refusing new sessions rather than letting the server thrash. Hitting it looks like "some people cannot open their form", not like a crash.
- **Max sessions per worker** defaults to **1** in the demo configuration, meaning one contribution session per worker process. That is the safest setting and the most expensive one; raising it trades isolation for density.

The demo store also runs contribution sessions **out of process**, recycling a worker after 25 sessions. Those settings are the ones to look at when someone asks how it behaves under load.

What to change when you scale up

At three forms	At three hundred
Assign individuals	Assign **groups**: one form per team, not per person, where ownership allows
Invitations on a daily clock	Same, but check the send volume; 300 invitations at 08:00 is a mail spike
Export on demand	Consider a **continuous** export so the model is not waiting on the whole population
Approvals one level up	Make sure one approver is not covering 300 forms
Reminders armed but unclocked	Actually schedule them, at scale, chase-ups are most of the process

Answering "will this cope with our headcount?"

The honest answer has three parts, and none of them is a single number:

1. **Here is how we measure it**: the memory profiling view, run against a template shaped like theirs.
2. **Here is how we prove it**: the stress-testing and analysis views, showing what a population of users experienced rather than a single-user timing.
3. **Here is the arithmetic**: the two ceilings above, and the sizing loop.

The number depends on your form, which is why measuring yours beats quoting anyone else's.

Related

- The campaign catalog · Update an input form
- Customer guide: Scale and size a contribution campaign, Profile and stress-test a campaign

Try out PowerPoint

how-to report author

"We rebuild the board deck by hand every month" is the problem this solves, and Timberline is the working proof. The deck contains no data, it is a template whose shapes are bound to Excel and refilled on every run.

Run it first

1. Open **Workspace ▸ Timberline ▸ Financial Reports ▸ Board Packet**.
2. Run it. It produces **one deck per region**, four decks, from one template and one workbook.
3. Open a deck. Every figure, table and chart came from `Timberline_Board_Pack.xlsx`; the title, the running footer and the headline strings came from cells in it too.

Board Packet with Division Slides exists precisely to show this: its second source workbook contributes one Revenue Review tab that repeats per market. Run it and compare: same template, but instead of four decks you get **one** high-level deck whose sales revenue slide repeats once per division. That difference is replication, not PowerPoint, see How parameters work.

Then open the template and see how

`Timberline_Board_Pack.pptx` is 8 slides, 16:9, every slide on a blank layout with explicit shapes. Three binding mechanisms are at work, and the deck uses all three.

1. Cell tokens in text

A text box whose content is `{SlideTitle}` or `{A45}` is replaced at run time with that named range or cell's value.

Token	Comes from	Where
`{SlideTitle}`	cell `F1` of the board pack's hidden data sheet	the cover slide's title
`{A45}`	the running footer line on each report sheet	every content slide
`{A5}`, `{A6}`, `{K1}`	headline and subtitle cells	slides 3, 6, 8

This is why the deck says "FY2026 Board Review, $ in millions" without anyone editing PowerPoint. The strings live in the workbook.

2. Linked shapes

Picture shapes carry a link naming the report, the worksheet, and the object to render into them. Three kinds are in use:

Link kind	Example	What it renders
named range	`Executive Summary / NetRevKPI`	that named range, as an image
cell range	`Outlook / I14:N25`	an explicit address range
chart	`Portfolio Mix / Chart 2`	that Excel chart

The deck links **21 named objects** across five worksheets: the six KPI tiles on the at-a-glance slide, the summary P&L, the earnings-bridge income statement, three Outlook tables, the plan assumptions, four charts, and two market-review tables from a second workbook.

Prefer named ranges to raw addresses. A named range survives someone inserting a row in the source workbook; `I14:N25` does not. Timberline's main deck binds mostly by named range, which is the pattern to copy.

3. Bookmarked slide sections

Each worksheet in the job carries a **parent bookmark name** that selects which slide section it feeds:

Job	Worksheet	Bookmark
Board Packet	the four report sheets	`Board Packet`
Divisional Packet	the four report sheets	`Board Deck`
Divisional Packet	`Revenue Review`	`Market Review`
Sales Rep Scorecard	`Rep Scorecard`	`Sales Rep Scorecards`

Because `Revenue Review` replicates **one page per item**, its bookmarked section repeats, one market slide per market value, inside one deck. That combination is how a fixed template produces a variable-length deck.

What the job controls

The board jobs all set the same PowerPoint options: **200 PPI**, **fit to width**, and `OFFICECOMPATIBLEEMF` image format. EMF stays crisp when a viewer scales a slide, at the cost of a larger file, a reasonable default for a board deck.

A job needs a template to emit a deck. Headless PowerPoint generation has nothing to build into otherwise. Note that *Divisional Packet* carries a template but currently outputs Excel and PDF only, so it will not produce a deck as configured.

Gotchas worth mentioning in a demo

- **Copying a linked shape breaks the link.** The binding travels with the shape's tag; paste it into another slide or deck and it stops updating.
- **One replicating worksheet per slide section.** Mapping several replicating sheets into one bookmarked section produces unpredictable ordering.
- **Raw-address links break when rows move.** Use named ranges.
- **Recipients can refresh decks themselves.** The business accounts hold the *PowerPoint End User* entitlement, so refreshing a deck is not an author-only action.

Related

- The report catalog · How parameters work
- Customer guide: PowerPoint reporting

Schedule jobs

how-to | report author | administrator

Timberline ships one schedule, deliberately switched off.

Property	Value
Name	**Board Packet Delivery**
State	**disabled**
When	`0 8 * * 4`, 08:00 every Thursday
Jobs	*Board Packet*, *Divisional Packet*
Hold for delivery	off
Parameter overrides	none

It is the demo's answer to "the board pack goes out every Thursday morning": two jobs on one schedule, so one entry drives the whole delivery rather than two schedules that can drift apart.

It ships disabled on purpose. A committed demo store that starts mailing on its own is a bad default. If you want to show a scheduled run, enable it deliberately, and disable it again afterwards, or it fires on the next Thursday.

Scheduling a job

1. Open the job, or go to the scheduling screen, and create a schedule.
2. Give it a **cron expression**. Reportworq uses **5-field cron**, minute, hour, day of month, month, day of week. `0 8 * * 4` is 08:00 on Thursdays.
3. **Add the jobs.** One schedule can carry several, which is the useful part: a delivery that spans three jobs stays together.
4. Optionally set **parameter overrides**, so the scheduled run uses different values from an ad-hoc run, a scheduled month-end run pinned to the closed period, for example. Timberline does not use this, but it is the reason parameter overrides exist.
5. Decide on **hold for delivery** (below).
6. Enable it.

Demo it safely: hold for delivery

Hold for delivery is the setting to reach for when demonstrating a schedule. The run happens on time, the output is produced, and delivery waits for a human to release it

from **Activity**.

That gives you a scheduled run you can talk over, "it ran at eight this morning, here is what it produced, and nothing has gone out yet", without committing to the send. It is also a genuinely good production pattern for a first month: schedule it, hold it, review it, release it, and only then let it go automatic.

Timberline's schedule has it off, so enable it before enabling the schedule if you want the safe path.

Campaign automation is a different thing

Do not conflate the two. **Job schedules run content on a clock. Campaign automation runs actions against a campaign**, and its triggers are richer.

The *Segment & Channel Budget Input* campaign carries ten automation tasks using four trigger shapes:

Trigger	Tasks	Behavior
Schedule: `0 8 * * *`	contributor invitation, approver invitation	invite daily at 08:00
Schedule, no cron set	the two reminders, stop campaign, auto archive	armed, not clocked
Form status change	contributor workflow, approver workflow, the `FixedCosts` export	fire the moment a form moves state
No trigger	the `Plan1` export	run on demand

The teaching point is that the *same action* means something different depending on when it fires. The `Plan1` export has no trigger and excludes unapproved data, it is the end-of-cycle export. The `FixedCosts` export fires on every form status change and *includes* unapproved data, it keeps the model warm while contributors are still working. See Export campaign data.

A caution for demo stores: the invitations fire at 08:00 daily, so a demo that wants an invitation *now* must trigger it rather than wait. And leave `auto_archive` disabled, enabling it would archive the campaign and refresh its forms, which destroys the demo state.

Related

- The report catalog · The campaign catalog
- Customer guide: Schedule a job