reportworq

AI & Integrations Guide

Enterprise reporting, distribution, and governed AI - product documentation

Version 6.0.0.0

Contents

AI & Integrations Guide

- Plan your integrations 4
- Source report providers 7
- The integrations hub 11
- Authentication providers 14
- AI in Reportworq 17
- Copilot, MCP, and trust paths 20
- The automation API 24
- Secure content before you expose it to AI 27
- Connect a data source 30
- Microsoft 365 OAuth setup 34
- Connect an AI provider 38
- Microsoft 365 Copilot agent 41
- Run a job by API 44
- Connector catalog 48
- The AI assistant 53
- The MCP shim for desktop AI clients 55
- Turbo Integrator, Workato, and Task Scheduler 58
- Connect IBM Planning Analytics 61
- AI-generated insights in reports 66
- MCP audit 69
- Deploy the Office add-in 73
- Connect SQL, OLE DB, and ODBC 76
- The AI prompt store 81
- The Script Runner 84
- Connect Power BI and Analysis Services 88
- Script API reference 96
- Data models 104
- Example scripts 109

AI & Integrations Guide

Plan your integrations

explanation | administrator | integration user

This guide is for the person deciding **what Reportworq should connect to** and **on what terms**. That is a different job from getting the server installed, licensed, secured and backed up, which is the Administrator Guide.

The split is deliberate, because the two jobs are usually done by two different people at two different times:

	Administrator Guide	This guide
The question	Is the environment sound and ready to hand over?	What are we going to connect it to, and how do we govern it?
Covers	Install, licensing, upgrade, migration, accounts and entitlements, workspaces, server configuration, capacity and topology, operations, retention, auditing	Source report providers, data sources and connectors, authentication providers, AI providers, Copilot and MCP, the REST API, the Script Runner
Typical owner	IT or platform operations	The integration or implementation owner, often working with the business
When	Once, at install, then on a maintenance cadence	Continuously, as new systems come into scope

If you are reading this and the server does not exist yet, start in the Administrator Guide and come back.

The five decisions

Almost every Reportworq implementation comes down to the same five decisions, in roughly this order.

1. Where do report source files live

Reportworq will not render a workbook from an arbitrary path. Source files come from a configured **report provider**, and that list is a security boundary as much as a convenience. Deciding this early matters because it determines who can put a file somewhere Reportworq will read it.

See Source report providers.

2. What data are we actually connecting to

The connector list is long, and the honest answer is usually shorter than the wish list. Work out which systems are in scope for the first phase, which of them you have credentials and network access for, and which ones need a service account provisioned by someone else.

Two things to settle per source: **who owns the credential**, and **whether the connection is workspace-scoped or instance-wide**.

See Connect a data source and the Connector catalog.

3. How do people sign in

One authentication provider is active at a time. Switching later is not a free action: existing accounts stay bound to the previous provider and have to be re-established under the new one. Decide before you onboard users, not after.

See Authentication providers.

4. Are we turning on AI, and under whose terms

AI in Reportworq is opt-in and provider-agnostic. The decisions are which provider, whose subscription and data-handling terms apply, and which surfaces are enabled. The prompts themselves are editable and auditable, which matters if you have a review process.

See AI overview and Connect an AI provider.

5. Are we exposing Reportworq to AI clients, and how do we keep content safe

This is the decision that most often needs a written answer before anyone will sign off. Enabling the MCP server means an AI client can reach report content. The controls exist, but you have to choose them deliberately.

See Secure content before you expose it to AI.

Two more that come up later

Automation. Once jobs are running on a schedule, something upstream usually wants to trigger them instead, a Turbo Integrator process, a Workato recipe, a nightly ETL. That is the REST API. See REST API overview.

Extensibility. When the requirement is "the output must also do X before it is sent", that is the Script Runner. Scripts run on the Reportworq server, so decide who should be able to add one.

What good looks like

An implementation is in reasonable shape when you can answer these without checking:

- Every report source file comes from a provider on a list you approved.
- Every data connection has a named owner and a credential that is not a person's personal account.
- The authentication cutover plan includes how you get back in if it fails.
- If AI is on, someone has read the prompts that are being sent.
- If MCP is on, someone has decided what a reader is allowed to see through it, and tested it as a reader.

Related pages

- The integrations hub, the single screen where every connection is registered and tested.
- Administrator Guide: system requirements for the environment side.
- Accounts, groups and entitlements for who can do what once they are in.

Source report providers

explanation | administrator | integration user

A **report provider** is a configured location that supplies the source files a report renders: Excel workbooks and PowerPoint templates. When an author adds a source on the Reports step, Reportworq opens the **Select a Report** dialog, which browses a report provider as a folder tree with a file grid.

Providers are managed in **Settings, Integrations**, in the **Report Providers** family. See The integrations hub for the add, test, enable and delete mechanics that apply to every integration family.

They are a security boundary, not a convenience

This is the point that gets missed. Reportworq permits report content **only** from trusted locations defined as report providers. A job cannot be pointed at an arbitrary server path, so it cannot be based on an invalid source or aimed at a sensitive file outside the approved locations.

That makes "which locations do we register" an access-control decision, not a shortcut. Anyone who can write a file into a registered provider can get that file rendered by Reportworq. Scope provider roots as tightly as the reports actually need.

You may define an **unlimited** number of providers.

Provider types

Provider	What it reads	Writable from the dialog
Workspace files	Files stored in the Reportworq workspace itself, shown with the same folders, names and permissions as the workspace file browser	Yes
Network folder	A share or path reachable from the Reportworq server	Yes
SharePoint	A SharePoint document library	Yes
OneDrive	A OneDrive location	Yes
Google Drive	A Google Drive location	Yes
Box	A Box folder	**No, read-only**
IBM Planning Analytics application folders	Workbooks published in a Planning Analytics application folder	**No, read-only**

Read-only providers are browse-and-load only. The Select a Report dialog hides its upload, rename, delete and create-folder buttons for them, because the provider reports that it does not support those operations. This is why the dialog looks different depending on which provider you are in; it is not an inconsistency.

Do not confuse the Box report provider with the Box distributor. Reading source files *from* Box is read-only. Delivering output *to* Box is a separate distributor, and that one writes. The same is true for the other locations that appear in both lists.

Prerequisite: at least one provider must exist

Add a Report requires a configured report provider. On a fresh install with none registered, the action reports *"No report providers are configured"* and authoring cannot begin. Provider setup is a genuine prerequisite of the first report, so put it early in your implementation plan.

Working with a provider

Authors do not configure providers, they consume them:

1. On the Reports step, select **Add a Report**.
2. Navigate the **provider tree** on the left and pick a file from the **file grid**.

3. Confirm with **OK**.

Both Excel workbooks and PowerPoint templates appear in the provider, so either can be added as report source content.

On a writable provider, **Upload File(s)** brings a local file into the provider so it can be added as a source. On a read-only provider the file has to arrive by that system's own means, for example someone putting it in the Box folder.

Dynamic, parameter-driven source paths

A source file's path can embed parameter values, resolved beneath the provider root at run time. For example:

```
c:\temp\%param:BU% Business Unit.xlsx
Templates/%param:Region%/report.xlsx
```

One job then reads a different workbook per parameter combination, which is how a burst can render genuinely different templates per business unit rather than the same template with different filters.

A report using this shows a **lightning-bolt icon** in the Reports grid, with the tooltip *"This report uses a dynamic report path."* That icon is your signal that the source file is not fixed.

The resolved path is still confined beneath the provider root, so the security boundary holds.

Two operational traps

Disabling a provider breaks viewing and running. A disabled provider can no longer resolve its sources, so authors cannot browse its files **and** any job referencing a file from it fails to run. Disabling is not a soft action; treat it as a change with job-level impact.

Changing a provider's folder location may strand existing jobs. Jobs hold a reference to the file as it was resolved. If you move the root, expect to update the jobs that referenced the old path.

A note on "Reportworq Drive"

If you are coming from version 5, "Reportworq Drive" split into two distinct things in version 6:

- A **built-in provider** backed by the repository. It is **retired**: you can no longer create a new one, but existing instances stay resolvable so older jobs keep working. Disabling one does not delete its files, it only stops listing it.
- **Workspace file storage**, which is the replacement and is the provider you should use for new work. It is a thin layer over workspace file items, so the dialog shows the same folders, names and permissions as the workspace file browser, and uploads honor the configured file-extension allow-list.

Do not treat the two as interchangeable when planning a migration.

Related pages

- The integrations hub for registering, testing and enabling connections.
- Plan your integrations for where this decision sits in the sequence.
- Destinations for the write side, which is a separate list.
- Script Runner, whose `.csx` files are also loaded through a report provider and inherit its access control.

The integrations hub

how-to administrator

Settings, Integrations is the one surface for every external connection Reportworq makes. If Reportworq needs to reach a data source, deliver to a destination, call an AI model, read source files, or authenticate users, that connection is registered and tested here. It is reached from the **Integrations** card on the Settings landing.

Connections are grouped into six families:

Family	What it holds
Authentication	The sign-in providers. Exactly one is Active (the built-in Native provider by default).
Platform	The MCP Connection tool, the entry point for the MCP and Copilot access path (administrators).
AI Connections	AI providers: OpenAI, Azure OpenAI, AWS Bedrock, and Reportworq AI.
Datasources	Data connections: TM1 / Planning Analytics, SQL / OLE DB / ODBC, and the other connectors.
Distributors	Delivery targets: email, network folder, SharePoint, and the rest.
Report Providers	The source-file locations reports are authored from.

Each connection appears as a card with a name, an on/off status pill, and a context (three-dot) menu.

Note: the Authentication family used to be a Configuration tab. It moved here, so manage sign-in providers on this screen, not in Server configuration.

Before you begin

- You need administrator rights.
- Have the credentials or endpoints for the connection you are adding ready (server names, API keys, service-account details).

Add a connection

1. Open **Settings, Integrations**.
2. Select **Add integration**. A full-page catalog opens with a search box and grouped tiles.

3. Search for or scroll to the integration you want, then select **Add** on its tile.
4. Fill in the connection editor. **Save** stays disabled until you change something.
5. Where the connection supports it, select **Test connection**. A good connection reports **Success**.
6. Save.

Some integrations are singletons, they can be added only once. A singleton that is already configured (for example the active authentication provider) shows in the catalog as disabled with an **Already added** note and no Add button.

Edit, test, enable, or delete a connection

Open a card's context (three-dot) menu for its actions:

- **Edit** opens the connection editor. As in the add flow, Save enables only once the editor is dirty.
- **Test** validates the connection and reports success or the error it hit.
- **Enable / Disable** toggles whether Reportworq uses the connection, without deleting it.
- **Delete** removes the connection.

The Platform, MCP Connection tool card has no context menu, it is a tool rather than a stored connection, and opens a full-screen editor.

Test everything at once

Before a large scheduled batch, confirm every connection is reachable:

1. On the Integrations list, select **Test all connections**.
2. In the dialog, select **Start**. Reportworq tests each enabled, testable connection and reports the results.
3. Select **Close** when done.

Authentication providers have no test path, so they are not included. If no enabled testable connection exists, the action shows a "no enabled connections" notice instead.

Filter the list

The header **status filter** narrows the list to **All**, **Enabled**, or **Disabled**. On **All**, every family group renders even when empty. On **Enabled**, groups with no enabled connections are hidden and only on-state cards show. When your account can reach more than one workspace, workspace-scoped families (such as Datasources) show a workspace-scope note.

Where to go for each family's detail

The hub is where connections are registered. Each family's setup detail lives with the feature it powers:

- **AI Connections**, see Connect an AI provider.
- **Datasources**, see Connect a data source and the Connector catalog.
- **Distributors**, see Destinations.
- **Report Providers**, the source-file locations authors build reports from.
- **Platform, MCP Connection**, see Copilot, MCP, and trust paths.

Notes and limits

- Instance-wide operational settings (web server, performance, logging, license) are in Server configuration, not here.
- Changing the active authentication provider warns that a service restart is required.
- Some connections are singletons and can be added only once.

> **Going deeper.** For the full list of connectors and their per-connector authentication notes, see the Connector catalog.

Authentication providers

explanation | administrator | integration user

Reportworq authenticates every user before any shell mounts. **Exactly one authentication provider is active at a time**, chosen in **Settings, Integrations**, in the **Authentication** family.

This page is the decision: which provider, and what it costs to change your mind. The step-by-step activation procedure, the self-service password flows and the lockout recovery path are in Sign-in and SSO.

The providers

Provider	What it is	What you need
Native	Reportworq's own username and password store. The default.	Nothing external. Lockout and password-complexity rules are set on the provider's own editor.
Microsoft Entra ID	OIDC single sign-on against Entra (formerly Azure AD).	Client Id, Client Secret, Tenant Id, and a completed Azure app registration. See Microsoft 365 OAuth setup.
Google	OIDC single sign-on against Google.	Client Id, Client Secret, and optionally Tenant Id.
Custom OIDC	Any standards-compliant OIDC identity provider.	Client Id, Client Secret, Tenant Id, the **Authority URI**, each claim name (User-ID, Email, Role, Name), and one or more scopes.

Every OIDC client secret is stored encrypted at rest.

Why this is an early decision

Three facts make the choice hard to reverse, and all three are worth knowing before you onboard a single user.

Switching strands the accounts on the previous provider. Changing the active provider does not merely drop permissions. Existing accounts stay bound to the provider they were created under and become reachable again only if you switch back. Under the new provider, their entitlement and group grants have to be re-established.

A service restart is required. Changing the active provider takes effect only after the Reportworq service restarts. Until then the previous provider stays active, which means the change is not something you can validate instantly.

There is no Test action. Authentication providers are excluded from the integrations hub's connection tests, including **Test all connections**, because there is no test path for them. You find out whether the configuration is right by restarting and trying to sign in.

Plan the cutover before you make it

If you are moving from Native to an OIDC provider, particularly Entra:

1. **Provision your way back in first.** Supply an initial administrator email that already exists in the identity provider, and confirm it resolves there *before* you cut over. Because the switch strands the old accounts, that pre-provisioned administrator is how you get back in if the cutover goes wrong.
2. Enter the provider's fields and save.
3. Mark the provider **Active**.
4. Restart the service.
5. Sign in as the pre-provisioned administrator and re-establish entitlements and group grants.

If even the pre-provisioned account fails, there is an operator-driven recovery path that requires restarting the process with an environment variable set. That deliberate friction is the proof you control the server. It is documented in Sign-in and SSO.

Roles can come from the identity provider

Once an OIDC provider is active, group or role claims from the identity provider resolve to Reportworq entitlements. That means you can assign a user's role through identity-provider group membership rather than granting it locally in Reportworq.

This is usually the reason to move to SSO in the first place: joiners and leavers are handled where they are already handled. Decide the claim-to-entitlement mapping with whoever owns your identity groups, not afterwards.

For what those entitlements actually permit, see Accounts, groups and entitlements.

One account per person

Licenses are sold as named users, and each account is one person. Sharing a login between several people breaches the terms of service, and it also destroys the audit trail, because every action is recorded against the account that performed it. If several people need access, give each of them an account.

This matters more, not less, under SSO, where it can be tempting to create a shared service identity for a team.

The Office add-in

The same sign-in surface handles the Excel and Office add-in login. When the login carries an Office cloud host, a successful sign-in posts the token back to the add-in's dialog host rather than navigating the shell, so the task pane connects without a separate browser session. **HTTPS is required for the add-in.**

Related pages

- Sign-in and SSO for the activation procedure, the native self-service flows, and lockout recovery.
- Microsoft 365 OAuth setup for the Azure app registration.
- The integrations hub for where the Authentication family lives.
- Plan your integrations.

AI in Reportworq

explanation | administrator | integration user

Reportworq's approach to AI is deliberately different from pointing a model at a database. The idea is **governed AI**: AI works on top of the financial content your finance team has already built, validated, and approved, not on raw databases, ERP tables, or the general ledger. This page explains the model behind that choice and what it means for you as an administrator or integration user before you turn any AI feature on.

The problem with pointing AI at raw data

Most AI projects begin by connecting a model directly to an ERP, a data warehouse, or a database. That sounds direct, but it forces the model to do the hardest parts of finance on its own: understand the schema, understand the business logic, work out how calculations are defined, decide which numbers are correct, and then do the arithmetic. Every one of those steps adds complexity and raises the chance of a wrong or invented answer.

Finance already solved those problems. Your team knows which reports are correct, and it validates those numbers every close. The reports carry years of curated terminology, business logic, and presentation decisions.

Governed AI: AI on trusted content

Reportworq puts AI on top of that trusted, finance-approved content rather than the raw systems underneath it. Instead of asking a model to recreate financial logic, Reportworq exposes governed reports the model can read, so the AI analyzes trusted answers instead of trying to calculate them.

This is the same shift described in the product's "AI done differently" idea: rather than forcing AI to rediscover the business on every prompt, Reportworq teaches AI how the business already communicates. Report authors enrich reports with structured business context, purpose, audience, strengths, limitations, terminology, and caveats, in a report's AI Profile. See Create an AI Profile for a report.

The practical results are:

- **Lower hallucination.** The model reasons over answers finance already trusts, not schema it has to reverse-engineer.
- **Lower token use.** Curated context is smaller and cleaner than raw data dumps, so prompts are cheaper.

- **Higher trust.** Answers trace back to the same reports finance already stands behind.

Every question and every answer is auditable

Governance is not only about which data the model sees. Reportworq knows who the user is, what they are allowed to see, which financial assets are trusted, how those assets are secured, every question asked, and every answer returned. Every interaction is recorded. Administrators can review AI activity in the audit logs. See Audit logs.

Users see only what they are authorized to see

AI in Reportworq respects the same access boundaries as the rest of the platform. A user asking a question through an AI feature sees only content they are already entitled to see. A department manager who receives a governed P&L can ask about it and get authorized detail back, without ever being given direct access to the general ledger. The AI does not widen access; it works inside the user's existing permissions.

Where AI shows up in the product

AI in Reportworq is delivered through a small set of features, each covered in this section:

- **Connect an AI provider** is the one prerequisite that turns on every AI feature. Nothing below works until an administrator has added and enabled at least one provider connection.
- **AI-generated insights in reports** attaches model-written narrative to a distributed report, and covers the `RWAIINSIGHT` worksheet function that writes AI narrative into a workbook.
- **The AI assistant** covers the in-app assistant surfaces, which are present in the product but not yet switched on in the current build.
- **The AI prompt store** is where an administrator reviews and tunes the internal prompts Reportworq sends to your provider.

Who this section is for

This section is written for **administrators**, who configure providers, govern access, and review audit trails, and for **integration users**, who consume AI-authored content in delivered outputs and downstream tools. Report authors who want to enrich a report for AI should start with Create an AI Profile for a report.

> **Going deeper.** The single most important next step is connecting a model. See Connect an AI provider.

Copilot, MCP, and trust paths

MCP-03 explanation administrator integration user

This section is written for two audiences: **administrators** who expose Reportworq to AI clients and external systems and set the posture that governs them, and **integration users** who connect a client and consume the result. Read this overview first to place the pieces, then follow the task pages for the surface you are setting up.

The three integration surfaces

Reportworq presents its report catalog and outputs to AI and automation in three ways. They are separate front doors onto the same governed content, and which one you choose depends on the client.

Surface	What connects through it	Best for
Microsoft 365 Copilot agent	Microsoft 365 Copilot, in chat, Teams, and the Microsoft 365 app	Enterprises standardized on Copilot that want governed, natural-language access to distributed reports. See Microsoft 365 Copilot agent.
MCP for desktop AI clients	Desktop and IDE AI clients (Claude Desktop, Cursor, Cline) and native-HTTP clients (ChatGPT, Copilot Studio)	Power users and developers who want to query reports from the AI tool they already work in. See The MCP shim for desktop AI clients.
The automation and REST API	Any external process: an iPaaS, an ETL job, a planning-system script, an OS scheduler	Programmatic, non-conversational job execution driven by another system's event. See The automation API.

The first two, Copilot and MCP, share the same underlying **MCP server** and its tool surface, so an AI client sees the same catalog and the same permission trimming whichever it uses. The third is a distinct HTTP command surface for running and monitoring jobs, covered in the Automation API section.

MCP and Copilot are the same tools, different clients

Reportworq serves the Model Context Protocol (MCP) from a single in-process server. Every AI client, whether Microsoft 365 Copilot or a desktop client behind the shim, calls the same read-and-act tools: list the report catalog, list a report's runs, fetch an output or its markdown, take a governed preview, and email an output. Access is trimmed by the same content-service ACLs and secured lists that govern the web UI, so an AI client

is a new way to **ask**, not a new way to **see**. A user never reaches a report or output variation their Reportworq permissions do not allow.

Copilot reaches those tools with **no client-side connector**, over the CloudHub relay. Desktop clients that speak MCP over a launched process use the **stdio shim**. Native-HTTP clients (ChatGPT, Copilot Studio) connect directly over HTTPS with no connector at all.

The two trust paths

Whichever surface a client uses, its request reaches the on-prem Reportworq instance by one of two **trust paths**. This is the central decision for an administrator, because it determines what the customer owns and what has to be opened.

Direct on-prem

The customer exposes the MCP or API endpoint to callers through a customer-managed reverse proxy (for example NGINX, Caddy, or IIS ARR) at a public DNS name with a public-CA TLS certificate. Nothing about Reportworq brokers the traffic; the customer owns the DNS name, the certificate, the reverse-proxy configuration, and, for the identity path, outbound reachability to their identity provider.

Choose direct on-prem when a security posture forbids cloud relays, for example an air-gapped or strictly outbound-only deployment. Note that a self-signed certificate does not work for cloud-hosted callers such as Copilot Studio; the certificate must be public-CA (Let's Encrypt is fine). Reportworq does not generate the reverse-proxy configuration, so this path hands the DNS, TLS, and proxy plumbing to customer IT.

CloudHub-relayed (the recommended default)

The on-prem instance registers with Reportworq CloudHub and holds an **outbound** connection to it. CloudHub forwards each request down that existing connection. There is **no inbound firewall change** and no public endpoint on the customer's server, the instance only ever dials out. This is the default, and it is what the Microsoft 365 Copilot agent uses.

Choose CloudHub-relayed for a fast rollout with no firewall change, and for any cloud-hosted caller.

Dimension	Direct on-prem	CloudHub-relayed
Network ingress	Customer reverse proxy at a public DNS name	Outbound connection from on-prem to CloudHub
Inbound firewall change	Required	None (outbound only)
TLS certificate	Customer-owned, public-CA	Terminated at CloudHub
Customer-owned plumbing	DNS, TLS, reverse proxy, identity-provider outbound	CloudHub registration only
Recommended for	Air-gapped, strictly outbound-only, regulatory bar on relays	Default, and any cloud-hosted caller

Who reaches what: identity is the boundary

The two trust paths are about **how traffic arrives**; a separate mechanism decides **what a caller may see**.

- Under the **identity (OAuth) path**, each request carries the signed-in user's identity. Reportworq resolves that identity to a Reportworq account and runs the request as that user, scoped to exactly the workspaces, folders, and reports that user's permissions allow. For the relayed Copilot agent, identity, not network location, is the security boundary: a token is honored only if it resolves to a real, enabled Reportworq user on the instance it reaches.
- Under the **API-key path** (used by the stdio shim and the relay's injected token), the request runs as the instance's System scope, full-workspace by design, optionally narrowed to a named workspace. There is no per-user identity on this path, but content-service trimming still applies.

Posture: turning surfaces on and off independently

An administrator gates each ingress with an MCP-scoped channel switch, so the two paths can be enabled or disabled independently:

- The **Direct URL channel** governs direct callers, and enabling it also turns on the workspace's Local API.
- The **CloudHub relay channel** governs relayed callers, and enabling it also turns on the Cloud Connector.

An instance-wide access mode (Disabled, Key, or OAuth) sits above both. Because the channels are MCP-scoped, turning off the direct channel stops AI agents on that ingress **without** disabling the shared Local API or Cloud Connector features behind it. This is what lets an administrator, for example, lock down the direct API surface while a relayed

Copilot agent keeps working.

Every request that reaches a tool, and every request rejected at the gate, is recorded. See MCP audit.

> **Going deeper.** For the full trust-path model, the download-delivery rules for large outputs, and the per-ingress gating detail, this overview's companion pages carry the depth: start with Microsoft 365 Copilot agent and The MCP shim for desktop AI clients. For programmatic job execution, see The automation API.

The automation API

API-REST explanation administrator integration user

Reportworq exposes a small HTTP API that lets an external process trigger and monitor job execution without using the web UI or the built-in scheduler. It is the only supported way to drive a job from outside Reportworq, and it is the substrate under the pre-built Turbo Integrator, Workato, and Task Scheduler integrations.

This page is written for **administrators** who enable the API and mint tokens, and for **integration users** who build the automated caller. It explains the two delivery models and their key models. To make actual calls, see Run a job by API.

When to use it. Reach for the API when automation cannot be expressed as a preset schedule, or must be triggered by another system's event, for example regenerating a reporting pack the moment a nightly data load finishes. For time-based runs, prefer the native scheduler. The API is positioned as a complement to the scheduler, not a replacement for it.

Two ways to reach the same commands

The same command set is delivered two ways from the same server.

	Local (Network) REST API	Cloud API
Where the caller runs	On the LAN, directly against the server	Anywhere on the public internet
Endpoint	`http(s)://{server}:{port}/api/v1/{command}` (default port **8600**)	Relayed through the Azure-hosted CloudHub proxy
Credential	A per-server, **per-workspace access token**	A **global Cloud Connector token**, one per server
Workspace selection	Implicit, the token maps to its workspace	Explicit, a `?workspace={name}` querystring
Network exposure	Direct LAN reach to the server	**Outbound 443 only**, no inbound exposure

Both are versioned under `/api/v1/`. Which one you choose depends on where the caller lives: on the same network as the server (Local), or out on the internet (Cloud).

The two key models are opposite (the common bug)

This is the fact to internalize before wiring anything, because mixing the two mental models is the most common integration mistake.

- **Local API: a separate key per workspace.** There is no workspace parameter. The access token you present **is** the workspace selector, Reportworq matches it against each workspace and resolves scope from the match. A token minted in the wrong workspace silently targets that workspace's jobs.
- **Cloud API: one global key plus `?workspace=`.** A single Cloud Connector token covers the server, and you name the workspace with a `?workspace={name}` querystring, matched by name, case-insensitively.

The `?workspace=` querystring is **required once more than one workspace exists**, and a single-workspace instance may omit it (the relay defaults to the sole workspace). On a multi-workspace instance, omitting it does not default, the call is rejected with a "missing or unknown workspace" message.

Cloud needs only outbound 443

The Cloud API opens no inbound port. The server dials **out** to CloudHub over 443 and CloudHub forwards requests down that connection, so the Reportworq server itself never needs to be reachable from the internet. This is what makes the Cloud API attractive to security teams: automation from a SaaS caller without opening the firewall. Cloud calls are subject to short Azure maintenance windows, so a robust caller retries on a transient outage.

Enabling and gating

- **The API is a licensed capability.** The **API** entitlement on your license must be present, or an API command is refused with HTTP 400 and the message "REST API not available for this license". This is a hard block, distinct from both "The Reportworq API is disabled" (the per-workspace toggle) and "Invalid Access Token". The `version` and `license` endpoints, output downloads, the contribution upload preview, the MCP server, and CloudHub setup are deliberately not gated by it. See Licensing and entitlements.
- The **Local API is enabled per workspace.** Even with a valid token, if that workspace's API is disabled the call is refused with "The Reportworq API is disabled." That message is distinct from "Invalid Access Token", so if a correct token fails, check the enable toggle before assuming a bad key. Enable and regenerate the token in the admin API settings; see Server configuration.

- The **Cloud Connector must be enabled** for the Cloud API. Changing the Cloud token breaks existing automations built on the old one, so rotate deliberately.

Tokens should be shared only with the developers who build the callers.

What the commands cover

The API is a small, five-command surface, and two older commands are retired:

- `ping`, a liveness probe.
- `list`, the jobs available to run.
- `run`, execute a job, with optional parameter and distribution overrides, returning a job id immediately.
- `status`, the current state of a running job.
- `cancel`, terminate a running job.
- `info` and `history` are **deprecated and return 404**, not 405. They are deliberate deprecation stubs, so a client that treats 404 as "wrong URL" will misdiagnose them. Do not build against them.

The detail of each command, and how to filter a single run to a specific output type and destination, is in Run a job by API.

> **Going deeper.** To run and monitor a job, and to send one job's output as a PDF by email while another caller gets the same job as a PowerPoint by Slack, see Run a job by API. To launch jobs from a planning system, an iPaaS, or the OS scheduler, see Turbo Integrator, Workato, and Task Scheduler.

Secure content before you expose it to AI

explanation | administrator | integration user

Enabling the MCP server means an AI client can reach report content. Everything needed to control that exists, but none of it is automatic, and the defaults differ between the two access paths in a way that surprises people.

This page is the checklist to work through **before** you turn it on. The mechanics of each path are in Copilot, MCP and trust paths; this is the governance decision that sits on top.

The one fact that drives everything

Two paths reach the MCP server, and only one of them carries a user identity.

Path	Runs as	Sees
Identity (OAuth)	The signed-in user, resolved to a real Reportworq account	Exactly the workspaces, folders and reports that user's permissions allow
API key (the stdio shim, and the relay's injected token)	The instance's **System scope**	**Full workspace by design**, optionally narrowed to one named workspace

On the identity path, your existing permission model is the control, and it does the work you expect.

On the API-key path there is **no per-user identity at all**. The request is not "Alice asking"; it is "the instance asking". Content-service trimming still applies, but per-user permissions cannot, because there is no user. Anyone who holds the key sees what the instance can see, bounded only by the workspace narrowing you configure.

Treat an MCP API key as equivalent to a full-workspace read credential, and handle it accordingly.

Work through these before enabling

1. Decide which path you actually need

If your callers are people using Copilot or a signed-in AI client, use the identity path and let permissions do their job. Reach for the API-key path only when the caller genuinely is a machine with no user behind it, and then narrow it to a named workspace.

2. Decide what a reader is allowed to see

Reportworq's permission model already answers this for the portal. The question is whether the answer is still right when the same content is reachable conversationally, where a reader can ask broad questions rather than navigating to a specific report.

Go through the workspaces in scope and confirm the folder and item permissions say what you intend, rather than what has accumulated.

3. Decide which surfaces stay hidden

Content marked hidden from readers stays hidden through MCP as well as in the portal. That is the mechanism for "this exists but is not for general consumption". Use it deliberately rather than relying on obscurity.

4. Gate the ingresses independently

Each ingress has its own MCP-scoped channel switch, and an instance-wide access mode (**Disabled**, **Key**, or **OAuth**) sits above both:

- The **Direct URL channel** governs direct callers. Enabling it also turns on the workspace's Local API.
- The **CloudHub relay channel** governs relayed callers. Enabling it also turns on the Cloud Connector.

Because the channels are MCP-scoped, turning off the direct channel stops AI agents on that ingress **without** disabling the shared Local API or Cloud Connector features behind it. That is what lets you lock down the direct API surface while a relayed Copilot agent keeps working.

Start with the instance-wide mode set to the narrowest thing that meets the requirement, and open up from there.

5. Confirm the audit trail is where you expect

Every request that reaches a tool, **and every request rejected at the gate**, is recorded. The rejections matter as much as the successes: they are how you notice a client probing for content it should not have. See MCP audit.

Decide who reviews that log and how often, before you need it.

Verify as a reader, not as yourself

This is the step most often skipped, and it is the one that catches real mistakes.

Administrators see everything, so an administrator testing MCP learns nothing about what a reader can reach. Verify with an account that has the permissions of your least-privileged intended user, and ask it the broad questions you are worried about, not the narrow ones you designed for.

If the answer surprises you, fix the permissions, not the prompt.

A note on scope creep

MCP access tends to broaden quietly. A workspace gets added, a folder's permissions get loosened for an unrelated reason, a key issued for one integration gets reused for another. None of these are visible as an "MCP change", but each one widens what an AI client can reach.

Two habits keep it honest:

- **Re-run the reader verification** after any material permission change, not just after MCP changes.
- **Issue one key per integration**, so that revoking one does not mean breaking all of them, and so the audit log tells you which caller did what.

Related pages

- Copilot, MCP and trust paths for the full trust-path model.
- MCP audit for what gets recorded and where to read it.
- Microsoft 365 Copilot agent and The MCP shim for desktop AI clients for each ingress in detail.
- Accounts, groups and entitlements for the permission model this all rests on.
- Plan your integrations.

Connect a data source

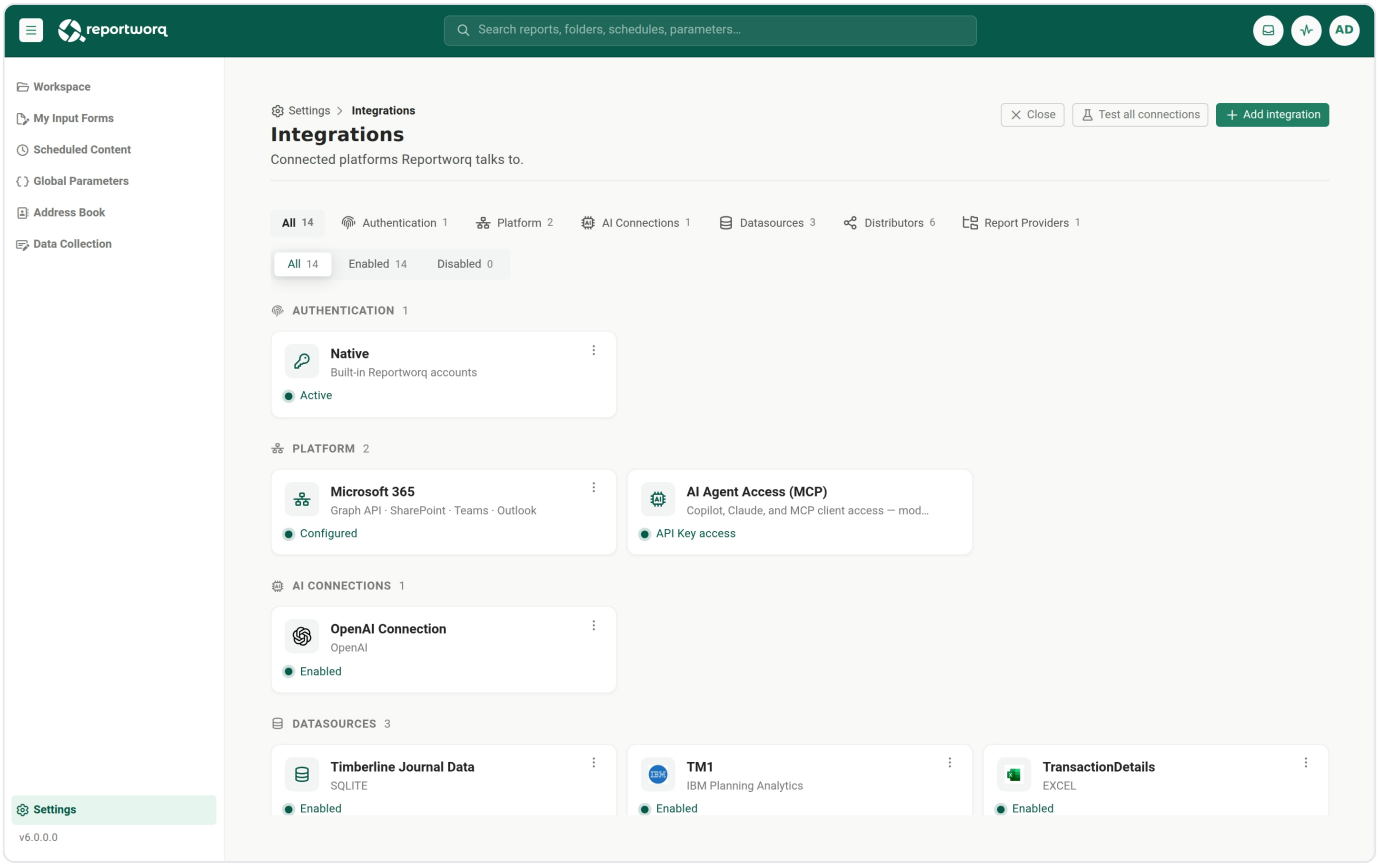
how-to administrator

Before a data model or a report can read data, the source system needs a **datasource connection** in Reportworq. A datasource is a named, saved set of connection settings and credentials that gives Reportworq access to a source, whether that is IBM Planning Analytics, SQL Server, a data warehouse, or a SaaS system. This guide covers adding a connector from the catalog, testing it, and managing it safely once jobs and forms depend on it.

Before you begin

- You are an administrator. Only administrators can add or configure datasource connections.
- The source system is reachable from the Reportworq server over the network, and any required driver or provider is installed on that server.
- You have valid credentials, API keys, or tokens for the source.
- You know the connector you need. If you are unsure, see the Connector catalog.

Add a datasource connection

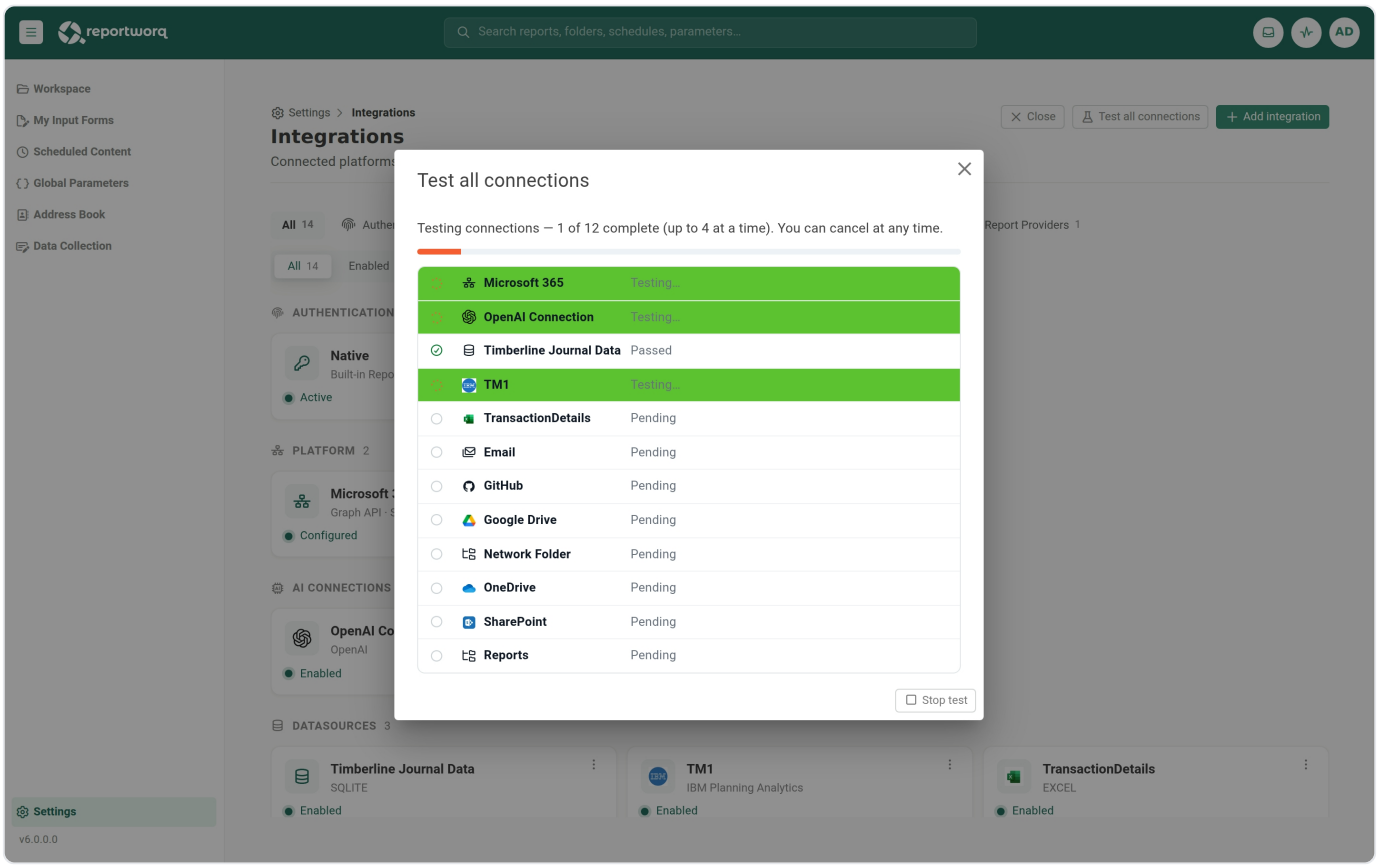


1. Go to **Settings** > **Integrations**. Connectors are grouped into families by their uppercase heading. Datasource connectors live under **DATASOURCES**.
2. Select **Add integration** in the header. A searchable catalog of connector tiles opens.
3. Find the connector you need and select **Add**. This creates the integration and opens its editor. A connector that can only exist once shows **Already added** if it is already present.
4. Fill in the connector-specific settings. Every connector asks for its own fields (host or server, credentials, cloud or on-premises mode, API keys, and so on). For the deeper connectors, see Connect IBM Planning Analytics, Connect SQL, OLE DB, and ODBC, and the Connector catalog.
5. Give the connection a clear **name**. The name is how models, parameters, and formulas refer to this datasource, so it must be unique across the instance. See the warning below on renaming.
6. Select the **Enable datasource connection** checkbox. This is the gate that makes the datasource usable. A disabled datasource cannot be selected or refreshed.
7. Select **Save**. Save stays disabled until you have changed something.

Test the connection

1. In the connector editor, or from the datasource card's `...` menu, select **Test connection**.
2. A reachable, correctly configured source returns **"Success."** If the test fails, re-check the host or server address, credentials, network reachability, and any required driver on the server.

To validate every configured integration at once, select **Test all connections** in the Integrations header. Reportworq then runs every connection test and reports the result of each in a dialog.



Manage a datasource

Each configured datasource appears as a card showing its name, an on/off pill, and a `...` menu with **Edit**, **Test**, **Enable** or **Disable**, and **Delete**. A status filter at the top of the list (All, Enabled, Disabled, with counts) narrows what you see.

Warning: editing, disabling, or deleting a live datasource has an org-wide blast radius. A datasource is referenced by name, and it is re-queried at run time by both Distribution Jobs and Contribution input forms. Changing its credentials, renaming it, disabling it, or removing it can silently break data refresh for every job and form bound to it. Treat any change to a live datasource as a production change: know what depends on it first.

Notes and limits

- **Datasource names must be unique.** Reportworq resolves a datasource by name, so a duplicate or a rename strands everything that referred to the old name.
- **A datasource is for reading, not delivering.** Delivery targets (email, network folders, SharePoint, and so on) are configured as **distributors**, a separate integrations family. Do not confuse the two.
- **When to disable rather than delete.** If you are only pausing a source, disable it. Deleting removes the credential record entirely and cannot be undone.

Going deeper. A datasource on its own is just a connection. To turn it into curated, reusable data for report authors, build a data model on top of it. See Data models.

Microsoft 365 OAuth setup

how-to administrator

To let Reportworq talk to Microsoft 365, you register **one** application in your tenant's Microsoft Entra ID (Azure AD) portal, then authenticate the running Reportworq instance to it. That single registration is the trust anchor for every Microsoft 365 capability, it is not per-feature.

When to use it. Any deployment that uses Entra sign-in, Microsoft-native distribution (email, SharePoint, or Teams), or email-based data collection.

> **One registration lights up everything, and breaks everything.** A single successful setup enables the **Microsoft Entra** authentication provider, the **SharePoint** report provider, the **Email, SharePoint, and Teams** distributors, and **email data collection** all at once. There is no per-feature registration. If you break the one registration (an expired secret, a wrong redirect URI), all of these fail together.

Before you start

- You need app-registration rights in your Azure / Entra tenant, and Administrator rights in Reportworq.
- Know the instance's advertised address (scheme, host, and port). The redirect URIs must match it exactly, including the `https` scheme, which Microsoft 365 requires. On a single-box install the advertised address is `https://localhost:8600`: 8600 is the default listen port, and a fresh Windows install serves HTTPS. If the instance runs behind a reverse proxy, see the note after the redirect-URI table.
- Decide the Microsoft account to authenticate with. Use a dedicated service account (see the warning at the end), not a person's mailbox.

Register the Azure application

1. In the Entra ID portal, create a new app registration. For directory type, choose **Accounts in this organizational directory only (single tenant)**.
2. Do **not** set a redirect URI on the creation screen. Redirect URIs are added later on the platform step.
3. From the registration **Overview**, record the **Application (client) ID** and the **Directory (tenant) ID**.

Create a client secret

1. Go to **Certificates & secrets ▸ New client secret**, give it a description and an expiry, and create it.
2. Record the **Secret value** shown immediately, not the Secret ID. The Secret ID cannot authenticate, and the value is unrecoverable once you leave the screen.
3. Note the expiry date. When the secret lapses, the whole Microsoft 365 integration stops working, so rotate it in Reportworq's Microsoft 365 settings before expiry. Set a calendar reminder.

Add the Web platform and redirect URIs

1. Go to **Authentication ▸ Add a platform** and choose **Web**. Any other platform type (SPA, mobile, desktop) silently fails the callback.

2. Enable **Access tokens**.

3. Add these three redirect URIs, all pointing back at your Reportworq server. Replace the default host below with your instance's actual advertised address if it is not a single-box `localhost:8600` install:

Redirect URI (default host)	Purpose
`https://localhost:8600/signin-oidc`	Entra sign-in callback
`https://localhost:8600/server/v0/get-office365-smtp-token`	Outlook / SMTP token callback
`https://localhost:8600/server/v0/get-office365-sharepoint-token`	Graph / SharePoint token callback

> **Behind a reverse proxy.** If Reportworq runs behind a TLS-terminating reverse proxy and its callback URLs come back as `http` instead of `https`, set `BehindReverseProxy` in `settings.json` so it honors the `X-Forwarded-Proto` header. See Deployment topology.

Authenticate Reportworq (two flows)

1. In Reportworq, go to **Administration ▸ Microsoft 365**. Enter the **Tenant ID**, **Client ID**, and the **Secret value**, then save.
2. In the **Graph API** area, select **Authenticate**, sign in as the Microsoft 365 account, tick **Consent on behalf of your organization** if prompted, and **Accept**. This grants the Graph-family scopes (Teams, SharePoint, Graph email, and the user

directory).
3. In the **Outlook API** area, select **Authenticate** again. This is a separate flow granting the legacy Outlook POP, IMAP, and SMTP scopes.

The two flows are distinct token grants: authenticating one does not authenticate the other, so complete both.

Scopes each flow requests

Reportworq requests a fixed set of scopes per flow. The Graph API flow requests the Microsoft Graph scopes below; the Outlook API flow requests the legacy Outlook scopes. Both flows also request `openid` and `offline_access`, which cover sign-in and silent token refresh. If your tenant requires admin consent, an administrator consents to these on the first Authenticate.

The Graph API flow requests these scopes:

Purpose	Scope
Read the signed-in user's profile	`https://graph.microsoft.com/User.Read`
Resolve recipient directory entries	`https://graph.microsoft.com/User.ReadBasic.All`
List teams	`https://graph.microsoft.com/Team.ReadBasic.All`
List channels	`https://graph.microsoft.com/Channel.ReadBasic.All`
Post to a channel	`https://graph.microsoft.com/ChannelMessage.Send`
Create a chat	`https://graph.microsoft.com/Chat.Create`
Post to a chat	`https://graph.microsoft.com/Chat.ReadWrite`
Attach files, including to SharePoint	`https://graph.microsoft.com/Files.ReadWrite.All`
Browse SharePoint sites and libraries	`https://graph.microsoft.com/Sites.Read.All`
Write output to a SharePoint library	`https://graph.microsoft.com/Sites.ReadWrite.All`
Send mail as the account	`https://graph.microsoft.com/Mail.Send`
Send on behalf of a shared mailbox	`https://graph.microsoft.com/Mail.Send.Shared`
Read mail for data collection	`https://graph.microsoft.com/Mail.Read`
Manage mail for data collection	`https://graph.microsoft.com/Mail.ReadWrite`

The Outlook API flow requests these scopes:

Purpose	Scope
IMAP mailbox access	`https://outlook.office.com/IMAP.AccessAsUser.All`
POP mailbox access	`https://outlook.office.com/POP.AccessAsUser.All`
SMTP send	`https://outlook.office.com/SMTP.Send`

Only the features you use need their scopes. If the deployment uses Entra sign-in and email only, you can remove the Teams and SharePoint scopes, as described next.

Tighten the scopes (optional)

- In **Administration ▸ Microsoft 365 ▸ Advanced Options**, delete unwanted scopes from the **Graph API** and **Microsoft 365 groups** scope lists. **Reset** restores the defaults. After editing a group's scopes, **Sign out** and **Authenticate** that group again so the new consent takes effect.
- To tighten the Azure side as well, in the registration under **API permissions** choose per scope **Remove permission** (the scope can be re-added later) or **Revoke admin consent** (blocks re-adding until an administrator re-consents).

Notes and limits

- The redirect URIs are host-specific. A server move or port change requires re-registering the URIs to the new advertised address.
- The client secret is time-bombed. Unmonitored expiry silently breaks every Microsoft 365 feature.
- The platform must be **Web**.

> **Use a dedicated service account.** The Microsoft account you authenticate with is, by default, also the Email distributor's sending account. Every burst email goes out as that identity. If you use a person's mailbox, all outbound mail appears to come from that person and breaks when they leave. Register with a dedicated Microsoft 365 service account instead.

> **Going deeper.** This registration is what enables the Microsoft Entra authentication provider (Sign-in and SSO) and the Microsoft email, SharePoint, and Teams destinations (Destinations).

Connect an AI provider

how-to | administrator | integration user

Reportworq's AI features do not embed a fixed model. Instead, an administrator configures one or more **AI provider connections**, each of which names a provider, holds its credentials and model selection, and carries an **Enabled** flag. Every AI feature, such as insights and the `RWAIINSIGHT` function, then resolves a model from the set of enabled connections. Configuring at least one enabled connection is the single prerequisite for all AI in the product.

When to use it. Do this once, during setup, before authors build AI-narrated reports or use the `RWAIINSIGHT` function. With no enabled connection, the AI features stay hidden or cannot run.

Before you start

- **Administrator access** to Settings, in the AI Connections family of the integrations hub. See The integrations hub.
- **Valid provider credentials** for the provider you intend to use. The one exception is **Reportworq AI**, which uses your license entitlement rather than an API key.
- AI connections are **workspace-scoped**: each workspace has its own connections.

Add and enable a connection

1. Go to **Settings ▸ Integrations** and choose **Add integration**.
2. In the **AI Connections** group, pick the provider tile. A new connection is created **disabled**, with provider defaults, and its editor opens.
3. Set a **Name**. This is how authors pick the connection later, and how the `RWAIINSIGHT` function resolves it, so make it recognizable.
4. Fill in the provider's required fields (see the table below).
5. Choose **Test connection**. Reportworq calls the provider and reports success or the provider's own error message.
6. Turn on **Enable provider**, then **Save**. Only **enabled** connections are used by downstream features. Save stays disabled until you have changed something.

Provider reference

Provider	Required fields	Notable options and defaults
OpenAI	API Key, Model Id	Optional Organization Id. Default Temperature 0.7, Default Max Tokens 1024. Model Id defaults to `gpt-5.1`. Temperature range 0-2.
Azure OpenAI	Endpoint, Deployment Name, API Key	Default Temperature 0.7, Default Max Tokens 1024. Temperature range 0-2. Create the Azure OpenAI resource and deploy a model first, then record the Deployment Name, API Key, and Endpoint.
Amazon Bedrock	Access Key Id, Secret Access Key, Region, Model Family, Model Id	Optional Session Token. Default Temperature 0.5, Default Max Tokens 1024. Temperature range 0-1. The Model Family dropdown currently offers **Anthropic only**. Region is the source region for request origination. Supply a Session Token only if your organization's security policy requires one.
Reportworq AI	Model Tier	No API key, access is brokered through your license with managed token quotas. Shows a Token Usage and Quota panel with a **Check now** action. Available only when your license grants the hosted-AI entitlement.

Notes and limits

- **OpenAI silently falls back to `gpt-4o-mini` when Model Id is blank.** A new OpenAI connection defaults its Model Id to `gpt-5.1`, but if the stored Model Id is ever cleared to nothing, the connection does not error, it quietly runs on `gpt-4o-mini` at call time. That can surprise anyone reconciling model choice or billing against what the editor appears to show, so always keep an explicit Model Id set.
- **Temperature ranges are imposed by the provider, not clamped in the form.** The temperature box accepts any number; an out-of-range value is not blocked when you save, but the provider's API rejects it at call time. Stay within the ranges above (OpenAI and Azure OpenAI 0-2, Bedrock 0-1).
- **Some providers are preview-gated or off in the current build.**
  - **Amazon Bedrock** currently exposes only the **Anthropic** model family in the editor.
  - **Azure AI Foundry** exists in the code but is **not registered**, so it does not appear as a choice in the current build. Do not plan around it.
- **Credentials are masked.** API keys and AWS secret and session keys render in a masked field with autofill disabled, and are stored through the AI connection content store.

- **Reportworq AI metering.** The hosted connection shows monthly quota, used this month, remaining, this session, lifetime total, and the reset date, with a **Check now** action. Other providers are not metered centrally.

> **Going deeper.** Every AI call made through these connections is recorded. See Audit logs. The internal prompts sent to the provider are editable in The AI prompt store.

Microsoft 365 Copilot agent

MCP-01 how-to administrator integration user

The Reportworq Microsoft 365 Copilot agent lets Copilot users ask about their reports in natural language, in chat, Teams, and the Microsoft 365 app, without leaving Copilot. The agent reaches your on-prem instance over the outbound-only CloudHub relay, and every request runs as the signed-in user, scoped to the reports, folders, and workspaces that user is allowed to see.

This is a two-role task. A **Reportworq administrator** generates the agent package. A **Microsoft Entra Global Administrator** grants one-time consent and deploys it to users.

When to use it. Reach for the Copilot agent when your organization has standardized on Microsoft 365 Copilot and wants governed, identity-scoped access to distributed reports, and when you cannot or do not want to expose an inbound endpoint. Desktop AI clients such as Claude Desktop use the MCP shim instead, and air-gapped deployments use direct on-prem MCP. See Copilot, MCP, and trust paths.

Before you begin

- The on-prem instance is registered with CloudHub (**Enable Cloud API** / the Cloud Connector is on). See Copilot, MCP, and trust paths.
- The instance is in **OAuth** access mode with the CloudHub relay channel enabled and a Cloud API key configured. The Copilot recipe only appears once these are set, because the production Copilot experience is per-user identity.
- You have the **Cloud API URL** (the CloudHub base URL) and the **Cloud API key** (the CloudHub connector key) to hand.
- A **Microsoft Entra Global Administrator** is available for one-time consent and deployment, and agent users have **Microsoft 365 Copilot** licenses.

Generate the agent package (administrator)

1. Open **Settings ▸ AI Agent Access (MCP)**.
2. Expand the **Connect: Microsoft 365 Copilot** recipe.
3. Fill in the package fields. Sensible defaults are supplied for each:

Field	Notes
Agent name	Defaults to "Reportworq". Shown to Copilot users.
Description	A short and a long description; defaults supplied.
Instructions	Guidance the agent follows. Reportworq appends a fixed response-style policy that always wins.
Conversation starters	Up to six suggested prompts, for example "What reports are available?". Four defaults supplied.
Color and outline icons	Defaults shipped; if you replace them, match the exact required dimensions.
Cloud API URL	The CloudHub base URL. Required.
API key	The CloudHub connector key. This is a routing key, not a workspace selector.

4. Choose **Download** to get the agent package (`.zip`) and note the **admin-consent link**.

5. Hand the `.zip` and the consent link to your tenant's Microsoft Entra Global Administrator.

The agent's identity is stable across API-key rotations, so a later re-download uploads as an **update**, not a duplicate. Regenerate and re-upload the package after a Reportworq upgrade that changes the tool surface, because Microsoft 365 captures the tool list at deployment time.

Deploy the agent in Microsoft 365 (Entra Global Administrator)

1. Grant one-time admin consent using the link supplied by the Reportworq administrator. The Reportworq Copilot application is a multi-tenant Entra app, so it works across tenants with tenant consent and no per-customer registration.
2. Deploy the package to users or groups. In the Microsoft 365 admin center, go to **Settings ▸ Integrated apps** and upload the agent `.zip`, then assign the recipient users or groups.
3. Consider a staged rollout: deploy to a small test group first, validate against Reportworq, then widen.

A Global Administrator can revoke the application at any time under Entra **Enterprise applications**.

Use the agent (integration user)

The first time they use the agent, a Copilot user signs in with their Microsoft 365 work account. From then on, results are automatically limited to their Reportworq permissions. They can, for example:

- Ask what reports are available.
- Ask for the figures in the latest quarterly forecast, answered from the report's content.
- Ask the agent to email a report in a chosen format, confirming the recipient first.
- Ask it to compare two runs of a report, quoting the source values from each.

Notes and limits

- **Downloads of large outputs go by identity-gated link or email.** A small output is returned inline in the conversation. A large one is handed a CloudHub download link that only the user it was issued to can retrieve. If your policy keeps download links off, oversized outputs are emailed instead and small ones still return inline.
- **The package holds no standing credentials.** Entra issues short-lived per-user tokens; a token reaches a given instance only over that instance's own relay connection.
- **Regenerate after a tool-surface upgrade.** Microsoft 365 captures the function list at deployment time, so a Reportworq upgrade that adds or changes tools requires a re-download and re-upload.

> **Going deeper.** For the outbound-only relay model, the identity-versus-network security boundary, and how the same tools reach desktop clients, see Copilot, MCP, and trust paths. Every agent request is recorded; see MCP audit.

Run a job by API

API-REST | how-to | administrator | integration user

This page shows how to drive job execution with the `/api/v1` commands: check the server is up, list the jobs you can run, run one, poll its status, and cancel it. It then shows the depth trick that makes the API worth reaching for, filtering a single run down to one output type and one destination, so the same job delivers a PDF by email to one caller and a PowerPoint by Slack to another.

For the two delivery models and their key models, read The automation API first.

Before you begin

- The target job is authored and appears in the job list.
- You have a token. On the **Local API** it is the workspace **access token**, minted in the admin API settings; on the **Cloud API** it is the Cloud Connector token. See The automation API.
- On the Local API the token also selects the workspace, so use the token minted in the workspace whose jobs you want. On the Cloud API, add `?workspace={name}` once more than one workspace exists.

Authenticate

Present the token as an `accessToken` header on every call:

```
accessToken: <your-token>
```

You may instead pass it as an `?accessToken=<token>` querystring, but the header form is preferred, the querystring form is less secure. On the Local API the access token **is** the workspace selector, there is no separate workspace parameter.

The commands

Command	Method	Body or query	Returns
`ping`	GET or POST	none	`{"message":"Pong"}`
`list`	GET or POST	`?includePath=true` (optional)	The jobs you can run; full paths when `includePath` is set
`run`	POST	a job specification (below)	a message plus a `jobId`
`status`	POST	`{"jobId":"..."}`	the job's state, `progress` (0 to 100), and `logReport`
`cancel`	POST	`{"jobId":"..."}`	`{"message":"Success"}`

`info` and `history` are deprecated and return 404. Do not build against them.

Run and poll

1. **`ping`** to confirm the endpoint and token are good.
2. **`list`** to get the exact job name or path.
3. **`run`** with a job specification. The minimum is the job `Name`; the call returns a `jobId` immediately (it is fire-and-forget).
4. **`status`** with that `jobId`, polling until the job completes. `progress` runs 0 to 100 and `logReport` carries the run log.
5. **`cancel`** with the `jobId` if you need to stop it. A canceled job is granted a 60-second grace period to finish and release memory before it is force-stopped.

A minimal run body:

```
{
  "Name": "Reports/Monthly/Board Pack"
}
```

Override parameters for a run

The run body can replace a job's authored parameter values for that one run, one entry per named parameter, without editing the job. Each override names the parameter and carries a typed value, for example a delimited `list`, an IBM Planning Analytics `subset` or `mdx`, an `mdxrepeater` that runs once per list item, a `sql` value list, or a Workday `adaptive` burst. Each type has its own required fields (for example a `subset` needs the server and dimension, a `sql` value list needs the connection name).

Filter a run to one output type and one destination

A Reportworq job can be authored to produce several formats and send to several destinations. On a single run, the caller can **narrow** that down without touching the job, using two override lists on the run body:

- **`AllowedOutputFileTypes`**, a subset of `excel`, `pdf`, `powerpoint`, `csv`.
- **`AllowedDestinationTypes`**, a subset of `file`, `email`, `sharepoint`, `sheets`, `slack`, `teams`.

Only the listed formats and destinations fire for that run. There is also a `ConvertPowerPointToPdf` flag to force a PowerPoint deck to PDF for the one run.

When to use it, the worked example. You have one authored job, "Board Pack", that can produce both a PDF and a PowerPoint and can send by both email and Slack. Two different callers drive it two different ways from the very same job:

Caller A, a finance mailer, wants the PDF by email:

```
{
  "Name": "Reports/Monthly/Board Pack",
  "AllowedOutputFileTypes": ["pdf"],
  "AllowedDestinationTypes": ["email"]
}
```

Caller B, a team-channel bot, wants the PowerPoint by Slack:

```
{
  "Name": "Reports/Monthly/Board Pack",
  "AllowedOutputFileTypes": ["powerpoint"],
  "AllowedDestinationTypes": ["slack"]
}
```

One authored job, two callers, two entirely different deliveries, with no duplicate job to maintain. This is the reason to constrain a run at call time rather than authoring a separate job per audience.

Download the produced file

After `run` and `status`, an integration can fetch the produced output over a token-authenticated download URL, so the caller can hand the file downstream (an approval step, a ticket, a notification) rather than wait for a distributor to deliver it.

Notes and limits

- **The API-override parameter types are a subset** of the fuller authoring parameter catalog. Pivot-table, SmartView, Anaplan, and PA-dimension authoring types are not distinct API override types.
- **Cancellation is not instant.** A canceled job gets a 60-second grace period before force-stop.
- **Three different refusals look similar.** "REST API not available for this license" (HTTP 400) means the **API** capability is missing from the license; "The Reportworq API is disabled" means the workspace's API toggle is off; "Invalid Access Token" means the key is wrong. Check them in that order.

> **Going deeper.** To launch these runs from IBM Planning Analytics Turbo Integrator, from Workato, or from Windows Task Scheduler, and for the PowerShell hang fix, see Turbo Integrator, Workato, and Task Scheduler.

Connector catalog

reference administrator

This is the overview of every datasource connector Reportworq can connect to. Each is added and tested the same way, in **Settings** > **Integrations** > **DATASOURCES**; see Connect a data source. The tables below give the category, the authentication approach, and the traps that most often block a connection.

The connectors that need the most setup have their own detailed pages, with screenshots and the full field-by-field walkthrough. Use this catalog to find the connector you need, then follow its detailed page:

- Connect IBM Planning Analytics for TM1 and Planning Analytics (self-hosted, Cloud (IBM SoftLayer), and Cloud (MCSP)).
- Connect SQL, OLE DB, and ODBC for relational databases, SQLite, and the CData bridged sources.

This catalog covers **datasource** connectors only, the sources Reportworq reads data from. **AI provider connections** (OpenAI, Azure OpenAI, and the other model providers, with their API keys) are a separate integrations family; to configure one, see Connect an AI provider.

OLAP and planning sources

Connector	Category	Auth and notes
IBM Planning Analytics (TM1)	OLAP / planning	Self-hosted (Native, Windows Integrated, or CAM), Cloud (IBM SoftLayer), or Cloud (MCSP). Some setups require config from IBM (CAM namespace, gateway URL, Welcome Kit user). See Connect IBM Planning Analytics.
Power BI / SSAS (ADOMD)	OLAP (native ADOMD; DAX, MDX, DMV)	Password, OAuth service principal, or OAuth interactive. See the Adomd traps below.
Workday Adaptive Planning	Planning API	Basic credentials, Adaptive Token, or Workday OAuth (5 fields). Metadata cache defaults to 24h; use **Refresh Metadata** after a structure change. Timeout defaults to 600s.
Anaplan	Planning API	Endpoint, Host, Authentication Endpoint (defaults to `auth.anaplan.com`), Username, Password. Credentials must match those used for the Anaplan Excel add-in.
Vena	Planning API	Basic credentials against the Vena API.
Pigment	Planning API	Export API Key, Metadata API Key, and Application. See the Pigment traps below.
Oracle (EPM Cloud / Smart View)	EPM Cloud REST, Basic auth	Host (`<env>-<id>.epm.<region>.oraclecloud.com`), tenant-prefixed Username (`<tenant>.<user>`), Password. This is **not** a relational Oracle database, see the trap below.

Relational databases

Connector	Category	Auth and notes
SQL Server	Relational	Server, database, credentials. See Connect SQL, OLE DB, and ODBC.
OLE DB	Relational (generic OLE DB provider)	Freeform connection string; keep credentials out of it with `%USERNAME%` and `%PASSWORD%`.
ODBC	Relational (generic ODBC driver)	Freeform connection string; 64-bit System DSN with the driver installed on the Reportworq server.
SQLite	Relational (file-backed)	Connection string is a file path, `Data Source=<path>`, resolved on the Reportworq server; LIMIT dialect.
Snowflake	Relational (pre-configured OLE DB)	Cloud data warehouse. Ships as a **pre-configured OLE DB** connector, **not** CData; queried through table or SQL mode.

CData bridged sources

CData builds drivers that make an API or SaaS system look like a relational database. Reportworq licenses and embeds them so that a system with no SQL surface of its own, Salesforce or HubSpot for example, can be queried with the same table-or-SQL experience as SQL Server. That is why these connectors behave like the relational ones and why their connection properties look like a database connection string.

Every CData source Reportworq ships is listed below. **Each connector name links to CData's own ADO.NET documentation for that driver**, which is the authoritative reference for its connection properties, supported tables, and authentication options. Reach for it whenever you need a property that is not on the Reportworq form. For the shared setup, see Connect CData sources.

Connector	Category	Auth and notes
Salesforce	CData (API to relational)	Basic auth: User, Password, and a **Security Token**. The token is not in the form, reset it in Salesforce (Profile > Settings > **Reset my Security Token**); it is emailed and appended to the password.
Jira	CData	Standard CData connection properties.
NetSuite	CData	Standard CData connection; can import a **saved search** schema.
Databricks	CData	Standard CData connection.
SAP HANA	CData	Standard CData connection.
HubSpot	CData	Standard CData connection.
Smartsheet	CData	Standard CData connection.
Dynamics 365	CData	Standard CData connection.
Excel	CData	An Excel workbook exposed as a tabular source.
Power BI XMLA	CData	AzureAD (user) or AzureServicePrincipal (unattended, OAuth grant type CLIENT); needs `Workspace=<name>`, an XMLA-exposed workspace, and an approved Azure app registration. Distinct from the native Adomd connector.
Azure Analysis Services	CData	Tabular; standard CData connection.

One source customers sometimes look for here is not a CData tile: **Snowflake** ships as a pre-configured OLE DB connector, see the relational table above.

Per-connector traps

- **Oracle is EPM Cloud / Smart View, not a relational database.** The Oracle connector reads Smart View report specs and re-queries the Oracle EPM Cloud REST API with Basic auth. It has no table or SQL surface. To reach a relational Oracle database, use ODBC, OLE DB, or a CData driver instead.
- **Salesforce needs a Security Token.** Basic auth alone is not enough. Reset the security token in Salesforce; it is emailed to you and appended to the password. The connection fails without it.
- **Pigment API-Export URLs are hard-validated.** An API-export model supplies a URL and payload, and the **URL must begin with** `https://pigment.app/api`. Any other URL is rejected at save with an error.
- **Pigment block imports.** Block imports cover four block types (Metric, Dimension List, Transaction List, Table), each needing its own data connection after import.
- **Re-importing creates a duplicate, not an update.** Re-importing an already-imported Pigment block, or re-selecting an already-imported relational table, creates an additional copy rather than updating the existing one. Edit the existing item in place to avoid silent model bloat.
- **Adomd (Power BI / SSAS) has three separate connection traps.** First, MFA-required accounts cannot use password mode; switch to a **service principal**. Second, an MDX query **must return a flat, tabular result** or it errors; use the MDX query type's crosstab render mode for matrix output. Third, XMLA is only exposed on **Power BI Premium, PPU, or Fabric** workspaces (not shared or Pro), with the XMLA endpoint set to Read or Read/Write, and for service principals the tenant-admin setting "Allow service principals to use Power BI APIs."
- **Workday Adaptive metadata is cached.** The dimensional-structure cache defaults to 24h freshness. If a source's structure changed, use **Refresh Metadata** before a run; save changes first.

Notes and limits

- Only administrators can add or configure datasource connectors.
- Datasource names must be unique, and editing, disabling, or deleting a live datasource breaks every job and form that refreshes from it. See Connect a data source.
- Writeback (data collection) paths, where available, are gated on a writeback license.

Going deeper. To turn any of these connectors into curated, reusable data for report authors, see Data models.

The AI assistant

explanation | administrator | integration user

Reportworq is building an in-app assistant, a contextual capability rather than a chat window, intended to help you understand, diagnose, and navigate the platform with evidence-backed, auditable answers. This page describes what that assistant is and, honestly, its current state: it is present in the product but not yet switched on for customers in this build.

Current state: not yet customer-available

The in-app assistant is not a customer-available capability in the current build. Its surface is gated off:

- **The Ask AI chat assistant** is a floating chat overlay whose agent can query your own reports and jobs through the same tools external AI clients use, scoped to your signed-in account. It ships wired off by default behind a build-time switch and cannot be turned on from an admin setting, so the **Ask AI** button does not appear in the header and the Reportworq Chat Bot card does not appear in the integrations hub.

Do not plan a rollout around this surface being available today. It is described here so you know what it is and how it is intended to work, not because it is on.

How the assistant is designed to behave

The assistant is built to a clear design contract:

- **Evidence before conclusion.** An answer shows its supporting evidence, not just a verdict.
- **Visible confidence.** A diagnosis states how confident it is.
- **Sources are inspectable.** Answers trace back to the content-store object, activity item, audit event, schedule, datasource, or job configuration behind them.
- **Mutations require approval.** The assistant is defined never to change configuration silently. Any change is staged as a before-and-after you must approve.

When enabled, the assistant will draw its model from a chosen enabled AI connection, and its internal instruction prompt will come from The AI prompt store.

The AI features you can use today

Several AI features are shipped and available now, once an administrator has connected a provider:

- **AI-generated insights in reports** attach model-written narrative to a distributed report, and cover the `RWAIINSIGHT` worksheet function that writes AI narrative into a workbook.
- **Connect an AI provider** is the prerequisite that turns those features on. Nothing model-backed works until an administrator has added and enabled at least one provider connection.

Notes and limits

All AI calls the product makes are recorded, including any made by the assistant when it is enabled. See Audit logs.

> **Going deeper.** To narrate reports rather than diagnose runs, use AI-generated insights in reports.

The MCP shim for desktop AI clients

MCP-02 | how-to | administrator | integration user

Desktop and IDE AI clients such as Claude Desktop, Cursor, and Cline speak MCP over a launched local process, while Reportworq serves MCP over HTTP. The **MCP shim** (`reportworq-mcp.exe`) is the small Windows connector that bridges the two: the client launches it, and it forwards the client's requests to Reportworq. Through the shim, the client sees the same governed report catalog and the same permission trimming as any other MCP client.

When to use it. Use the shim for a local AI client that launches an MCP server as a process on a Windows workstation. Native-HTTP clients such as **ChatGPT** and **Copilot Studio** need no connector, they connect directly over HTTPS, so for those the admin recipe gives you a URL-only snippet instead. Microsoft 365 Copilot uses the Copilot agent, not the shim.

Before you begin

- You are a Reportworq **administrator** to download the connector and generate the snippet; the resulting snippet and `.exe` are then handed to the **integration user** who runs the client.
- Decide the connection path:
  - **Direct on-prem** needs a workspace **Local API access token** and **Enable Local API** on.
  - **CloudHub-relayed** needs a registered CloudHub connection and its connector key. See Copilot, MCP, and trust paths.
- The workstation runs **Windows x64**. The v1 connector is Windows x64 only.

Download the connector and snippet (administrator)

1. Open **Settings ▸ AI Agent Access (MCP)**.
2. Expand the **Connect: MCP clients (Claude Desktop, Cursor, ChatGPT, Copilot Studio)** recipe.
3. Choose **Download Connector** to stream the bundled `reportworq-mcp.exe`.
4. Pick your **Client** and **Connection Path**. The recipe generates a copy-paste configuration snippet shaped for that client and path. Only the paths your channel switches allow are offered.
5. Hand the connector and the snippet to the integration user.

The connector also ships inside the main Reportworq installer, one copy per installed version. To update the binary on a workstation, re-download the current version from

this recipe.

Configure the client (integration user)

1. Place `reportworq-mcp.exe` somewhere on the workstation. Putting it on `PATH` is simplest; it needs no separate .NET runtime.

2. Paste the snippet into your AI client's `mcpServers` configuration block. The snippet's arguments differ by connection path:

Argument	Direct on-prem	CloudHub-relayed
`--server`	Your Reportworq URL	The CloudHub base URL
`--apikey`	Your workspace **Local API access token**	Omitted (the relay injects the workspace token)
`--path`	Not used	`/mcp/v1/<cloudhub-connector-key>`
`?workspace=<name>`	Not used	Appended to `--path` when more than one workspace exists; omitted for a single workspace

3. Restart the AI client so it launches the connector.

Test the connection and download readiness

Confirm the shim is wired before relying on it in the client, because a client that cannot reach Reportworq often shows a confusing "server crashed" or "no tools available" message rather than a clean transport error.

1. From a terminal, run the connector with your arguments and pipe an MCP `initialize` request to it. A single JSON-RPC line comes back with a server name of `reportworq-mcp`, which confirms the connector is framing correctly.
2. Follow with a `tools/list` request. A tool list confirms the upstream Reportworq instance is reachable and your credentials were accepted.
3. In the client, ask what reports are available. You should see the catalog trimmed to your permissions.

Notes and limits

- **Windows x64 only in v1.** There is no macOS or Linux build, and no standalone installer, the connector ships inside the main Reportworq installer.
- **API-key path, no per-user identity.** The shim authenticates with an API key, so it runs in the full-workspace System scope. Secured-list and folder trimming still apply, but there is no per-user identity on this path. For an identity-scoped, per-user

experience, use a native-HTTP OAuth client or the Copilot agent.

- **The direct path needs Enable Local API on.** With it off, the direct path is refused (as is a disabled Direct URL channel). The CloudHub path is gated separately by the relay channel and Cloud Connector.

> **Going deeper.** For the two ingress models the shim can target and how each is gated, see Copilot, MCP, and trust paths. Shim tool calls are recorded with a client and transport value; see MCP audit.

Turbo Integrator, Workato, and Task Scheduler

API-REST | how-to | administrator | integration user

The automation API is the substrate under three ready-made ways to launch a job from outside Reportworq: an IBM Planning Analytics Turbo Integrator (TI) process, a Workato recipe, and a Windows Task Scheduler task. This page covers each, and the one PowerShell setting that keeps a script-launched job from appearing to hang.

When to use it. Launch a job this way when the trigger is another system's event rather than a clock, for example a TI process that has just finished loading actuals and should regenerate the reporting pack on data-ready. If you previously used **Cube Monitoring** to launch jobs from Planning Analytics, note that Cube Monitoring is **deprecated**, and the REST-API Turbo Integrator described here is its replacement, it now also reaches IBM Planning Analytics Cloud through the Cloud Connector relay.

Before you begin

- The target job is authored and listable through the API.
- You have the right token for the delivery model: a workspace **access token** for the Local API, or the **Cloud Connector token** for the Cloud API. See The automation API.
- For a Cloud call from Planning Analytics Cloud, the on-prem instance is registered with CloudHub (the Cloud Connector is enabled), and you have the workspace name for `?workspace=`.

Launch a job from IBM Planning Analytics Turbo Integrator

Reportworq ships sample TI processes you adapt rather than write from scratch. They live in the TM1 plugin's `TI_Scripts` folder and include:

- **Rest API Run Job Sample**, and a **Cloud** variant that reaches Planning Analytics Cloud through the Cloud Connector relay.
- **Add Queued Job Sample** and **Add Queued Job For Workspace Sample**, for queued execution.
- A **PowerShell prolog** variant used by the Cloud sample.

To use one:

1. Import the sample TI process that matches your delivery model (Local or Cloud) into your Planning Analytics model.

2. Set the Reportworq server address (or the CloudHub base URL), the token, and the target job name.
3. For a Cloud call on a multi-workspace instance, set the `?workspace={name}` querystring.
4. Call the process where your model's data load finishes, so the reporting pack regenerates on data-ready.

The TI process calls the REST `run` command described in Run a job by API, so the same parameter and distribution overrides are available from TI.

Launch a job from Workato

The Reportworq Workato connector wraps the same command set as recipe actions:

- **Check Server Status**, returns true or false.
- **List Jobs**.
- **Run Asynchronous**, starts a job and returns its job id.
- **Run Synchronous**, starts a job and polls to completion, up to a `Timeout` in seconds.
- **Get Job Status** and **Cancel Job**.

The connector authenticates with the **Cloud API token**. Its `Parameters` and `Notification` inputs are semicolon-delimited. A typical recipe runs a job synchronously, waits for completion, then routes the result downstream to an approval, a ticket, or a notification.

Launch a job from Windows Task Scheduler

An OS task can issue a REST `run` on a schedule the native scheduler does not cover, most simply by having the task run a small PowerShell script that calls the API. If you script it in PowerShell, apply the fix below.

The PowerShell and TI hang fix

A job launched from a PowerShell script (including a Turbo Integrator PowerShell prolog) **delivers its reports correctly, but the script itself never returns**, it stalls after a successful run. The reports are never the problem, only the script's completion is. The cause is Windows `Invoke-WebRequest` trying to parse the server's HTML response.

Add this line near the top of the script:

```
$PSDefaultParameterValues['Invoke-WebRequest:UseBasicParsing'] = $true
```

This is now shipped by default in the generated Cloud TI PowerShell prolog, so newer generated scripts are already immune. It remains the fix for **hand-rolled or older** PowerShell scripts and older self-hosted run scripts. If a script-launched job delivers output but the launching script hangs, this is the setting to add.

Notes and limits

- **Cube Monitoring is deprecated.** Migrate job launches from Planning Analytics to the REST-API Turbo Integrator here.
- **The Cloud path needs only outbound 443.** A Planning Analytics Cloud model reaches your on-prem instance through the Cloud Connector relay with no inbound exposure. See The automation API.
- **Cloud calls are subject to short maintenance windows.** Build retries into an unattended caller.

> **Going deeper.** For the command surface these integrations call, parameter overrides, and filtering a run to a specific output type and destination, see Run a job by API. For choosing an authentication type when connecting to Planning Analytics, see Connect IBM Planning Analytics.

Connect IBM Planning Analytics

how-to administrator

The **IBM Planning Analytics** connector (TM1) connects Reportworq to a Planning Analytics server so its cubes, dimensions, subsets, views, and MDX can be modeled and reported. It supports self-hosted (on-premises) and cloud deployments. Getting the authentication right is the main task, so this page gives that section the depth it needs, then covers the deployment traps that most often reach support.

Before you begin

- You are an administrator, working in **Settings** > **Integrations** > **DATASOURCES**. See Connect a data source for the general add-and-test flow.
- You know which deployment you are connecting to: self-hosted, Cloud (IBM SoftLayer), or Cloud (MCSP).
- You have the credentials or key for that deployment. Some setups require you to request configuration from IBM first, see below.
- The account you connect with has at least read permission to refresh data. Data collection (writeback) additionally needs write on the target cubes and dimensions.
- Best practice: connect with a **dedicated Reportworq service account** provisioned at **data-admin** level, not an individual person's login. A shared service account keeps the connection working when people change roles or leave, and scopes access to what Reportworq needs. Full server-admin permission is required only once, to install the REST API Turbo Integrator objects (see below), and can be removed afterward.

Choose the authentication type

The connector's authentication type has three deployment values, and each shows a different field set. Choose the one that matches your server, then choose how credentials are presented.

Self Hosted (on-premises)

Use this for a TM1 server you run yourself. Fields:

- **Database**, a combobox. The **Load from Admin Host** button can populate it from the Admin Host server list, auto-filling Address, Port, and SSL from each entry's REST port.
- **Address**, the server as an IP address, machine name, or fully qualified domain name.
- **Port**, in the range 1 to 65535.

- **Use SSL**, which must match the server.

For credentials, self-hosted supports three login modes. The single **Namespace / Domain** field carries whichever value the mode requires, and its label changes to tell you which:

- **Native.** Enter a **Username** and **Password**. Use this for TM1's own user directory.
- **Windows Integrated login.** Select **Use Integrated Login**. The credential field's label becomes **Domain** and authentication is mapped to Active Directory. Use this for a domain-authenticated (WIA) server. Integrated login additionally requires SPNs registered in Active Directory for the FQDN, machine-name, and IP variants of the server address.
- **Cognos Access Manager (CAM).** Leave integrated login off and enter the CAM **Namespace** along with the username and password. Use this when the server authenticates through Cognos / CAM.

When to use which: match the mode to how the TM1 server itself authenticates. If TM1 is native, use native; if it is AD-mapped, use Windows Integrated; if it is fronted by Cognos, use CAM with the namespace.

Cloud (IBM SoftLayer)

Use this for Planning Analytics on IBM Cloud (SoftLayer). Fields:

- **Database**, which defaults to `tm1`.
- **IBM Cloud URL**, for example `mycompanyprod.planning-analytics.ibmcloud.com`.
- **Username** and **Password**.
- **Namespace**, which defaults to `LDAP`.

Cloud (IBM SoftLayer) needs an IBM **Welcome Kit non-interactive user account**. If element names contain special characters such as a backslash, the REST API also needs the `nocanon` configuration.

Cloud (MCSP)

Use this for Planning Analytics on AWS. Fields:

- **Database**.
- **Data Center Url**, for example `us-east-2.aws.planninganalytics.ibm.com`.
- **TenantId**, for example `A1BCD2EFGHIJK`.
- **API Key**, entered in the Password field. It is used for distribution and, optionally, data collection.

You may need to request something from IBM support

Several of these values are not self-serve, so expect a step where you request configuration from IBM before the connection will succeed. Depending on your deployment, that can include a **CAM namespace**, an **Admin Host or gateway / Planning Analytics URL**, the **REST API enabled** on the server (a valid HTTP port in `tm1s.cfg`), an **IBM Welcome Kit non-interactive user** for Cloud (IBM SoftLayer), or the `nocanon` REST configuration for special characters. Plan for that lead time rather than treating a failed test as a Reportworq problem.

Separate credentials for data collection (writeback)

If the server license includes writeback, the editor shows a **Data Collection** section with **Use different credentials for data collection**, which reveals a parallel credential set (username and password or integrated login for self-hosted, API key for AWS MCSP). The whole section is hidden unless the license has writeback.

Test the connection

Select **Test connection**. A reachable, correctly configured server returns **"Success."** If it fails, re-check the address and port, the SSL toggle against the server, the authentication mode and namespace or domain, and, for integrated login, that the SPNs are registered.

Advanced Options you should know about

The connector's **Advanced Options** section carries a few settings worth understanding before you report against the server.

- **PAfE Calculation Mode** is on by default and does two distinct things. First, it controls how invalid elements render: on returns `#VALUE!` (Planning Analytics for Excel behavior), off returns blank (Perspectives behavior). Note the error is `#VALUE!`, not `#N/A`. Second, it flips how `SUBNM` and `DIMNM` resolve public versus private subsets when a subset is referenced by index. Set it to match how your source workbooks were authored.
- **Enable Concurrent Dynamic Report Processing** is on by default. It fetches the data for Dynamic Reports and Active Forms concurrently, across multiple threads, rather than one report at a time. This speeds up refresh of those report types. Leave it on unless you need to force single-threaded processing.
- **Enable Proportional Data Collection** is off by default and relates to consolidated-cell writeback.

Set up REST API job launching

Advanced Options also carries two actions for launching Reportworq jobs from TM1 itself, using a Turbo Integrator process:

- **Install REST API TI Process** creates the Turbo Integrator processes on the server.
- **Download REST API TI Powershell Script** downloads the PowerShell launcher (`RunApiJob.ps1`) that calls a job. This action is available for **Self Hosted** and **Cloud (IBM SoftLayer)** only.

Install does not overwrite an existing process. If a `.pro` Turbo Integrator process from a previous install is already on the server, selecting **Install REST API TI Process** again leaves it in place; it does not update it. To refresh a pre-existing process, delete it on the server first, then run **Install REST API TI Process** again. For the full launch workflow, see Turbo Integrator and automation.

Deployment traps to plan for

- **Disable other datasources on the same database.** When several Reportworq datasources point at the same TM1 or Planning Analytics database, disable all except the one the active job uses. Leaving multiples enabled makes the job bind against the wrong connection.
- **Cube Monitoring was removed in v6.** Earlier releases used a Cube Monitoring service to launch jobs, and it was historically the only launch option for Planning Analytics Cloud. It has been removed; there is no Cube Monitor setting in the connector. Launch jobs from TM1 through the **REST API Turbo Integrator** instead. The Cloud Connector relay now extends the REST API Turbo Integrator to Planning Analytics Cloud as well, so it is the single path for both self-hosted and cloud. Any Cube Monitoring objects left on a server from a previous release are inert and are not removed automatically. See Turbo Integrator and automation.
- **Integrated (CAM) auth can break after a Windows update.** Updates that further the deprecation of RC4-Kerberos can cause service accounts with no explicit Kerberos encryption types to be rejected, breaking the integrated-login / CAM path. If integrated auth breaks overnight after patching, enable AES128 and AES256 encryption types on each affected account, reset the account password so the change takes effect, then update the credentials in Cognos and in Reportworq.
- **Static Universal Reports are not supported.** The connector refreshes full or dynamic Universal Reports, but static ones are skipped with a warning in the log.
- **Explorations (PAfE) are not supported.** Reportworq does not refresh Planning Analytics for Excel Explorations. Use a supported report type instead: Custom Reports, Dynamic Reports or Active Forms, Quick Reports, or full Universal Reports.

Notes and limits

- Cross-tab formatting (Crosstab, Tuple Table, Single Row) applies to the Data View and MDX Data View query types.
- Admin permission on the server is needed only one time, to create the REST API Turbo Integrator objects, and can be removed afterward. Day to day, a data-admin service account with read access is sufficient (plus write on the target cubes and dimensions for data collection).

> **Going deeper.** To build a curated model over a Planning Analytics cube and publish it for report authors, see Data models. To launch Reportworq jobs from TM1 itself, see Turbo Integrator and automation.

AI-generated insights in reports

how-to | administrator | integration user

Reportworq can attach AI-authored narrative to a report so recipients get a written explanation alongside the numbers, with no manual writing or copy-paste. There are two mechanisms, and this page covers both:

- **Distribution-time insights**, an **AI Insight variable** produced on the Distribution step and embedded into the delivered output.
- **In-workbook insights**, the `RWAIINSIGHT` worksheet function, which writes narrative into the workbook during report generation.

Both call an enabled AI provider, so both require an AI provider connection first.

RWAIINSIGHT is the generative-AI function

`RWAIINSIGHT` (canonical name `RW.Tools.AIInsight`) is the **one** generative-AI worksheet function. Do not confuse it with the Workday Adaptive `AI*` functions (`AIEXTRACT`, `AITIER`, `AIINPUT`, and the rest). Those are a data connector; their `AI` prefix stands for Adaptive, and they have nothing to do with generative AI or large language models. For that distinction and the full Adaptive catalog, see Workday Adaptive functions.

Option A: distribution-time AI insights

Use this when the narrative belongs in the **delivery**, in the body of an email, Slack, or Teams message, and should be personalized per burst. The **AI Insight** variable is offered only in the message body, not in a subject line, an address field, or an output file name.

When to use it. A burst that sends each department its own P&L can attach a per-department variance commentary to each recipient's email, driven from the same parameters that drive the burst.

Before you start

- At least one **enabled AI provider connection**. The AI Generated Insights section is hidden if no connection exists.
- A distribution field that accepts variable tokens (for example, the email message body) to embed the insight into.

Steps

1. On the Job Editor's **Distribution** step, open the **AI Generated Insights** section.

2. Choose **Add** to create a named prompt. Names must be unique; prompts show as a tab strip you can rename or delete.
3. Set the **AI Model** to one of your enabled connections. Selecting a model applies that provider's default max tokens to the prompt.
4. Open **Settings** to set **Max Tokens** (minimum 10, maximum 1,000,000, default 2048) and, optionally, **Include AI-Generated Footer**, which appends a note that the content was AI-generated and names the model.
5. Write the **prompt body**. It accepts free text and Reportworq variable tokens, so it can reference parameter and job values, for example, "Summarize the largest month-over-month P&L variances for {{Department}}." Images are not allowed.
6. Insert the prompt as an **AI Insight variable** into a variable-aware field, such as the email body. At delivery time the prompt runs and its generated narrative is embedded there.

An AI Insight is embedded narrative, surfaced in Run History's per-output AI Insights section. It is **not** a standalone output file.

Option B: in-workbook insights with RWAIINSIGHT

Use this when the narrative belongs **inside** the workbook, for example, a commentary box that flows into a PowerPoint or PDF output.

`RWAIINSIGHT` sends a prompt, optionally with a data range, to a configured provider during report generation and writes the response into a named Excel **TextBox** or a target **cell**. It runs after data import, sorting, and suppression, so the model sees the fully expanded dataset.

Arguments:

#	Argument	Required	Default	Notes
1	`providerName`	Yes	none	Matches an enabled provider's Name or Provider Id, case-insensitive.
2	`destination`	Yes	none	A TextBox name (searched across all worksheets) or a cell reference (its top-left cell is used).
3	`prompt`	Yes	none	The prompt text sent to the provider.
4	`dataRange`	No	none	Appended to the prompt as a Markdown table.
5	`maxTokens`	No	1000	Capped by the provider's configured limit.

Example: `RW.Tools.AIInsight("OpenAI", "CommentaryBox", "Provide a one-paragraph executive summary of the month's P&L variances.", A1:C20, 500)` sends the range as a table and writes the narrative into a TextBox named `CommentaryBox`.

On success the formula cell is set to an empty string. If the provider is not found or disabled, or the provider returns an error, the cell shows `#VALUE!`.

Notes and limits

- Both mechanisms need an **enabled** AI provider connection; token limits are ultimately bounded by the provider.
- Distribution-time insights and `RWAIINSIGHT` are separate paths. Distribution insights embed into the delivery message and fields; `RWAIINSIGHT` writes into the workbook itself.
- The AI Insight variable cannot be nested inside another prompt body.

> **Going deeper.** For where AI insights sit among the other delivery formats, see Output formats.

MCP audit

ADMIN-05 reference administrator integration user

The **MCP Server Log** is Reportworq's audit trail of AI access to reports over MCP. It records every MCP tool call that reaches a tool (who saw what, when) and every request rejected at the authentication gate before any tool ran. It is what makes AI access to reports accountable and revocable, and it is central to the Copilot agent's "auditable and revocable" security claim.

The log is one of four tabs in **Settings ▸ Auditing** (System Log, Job Log, AI Interactions, and **MCP Server Log**). It audits the tool request-and-result exchange, not the AI model's conversation, that conversation is the AI Interactions log. Access to Auditing is administrator only; a **User** or an **Author** cannot reach it.

When to review it. Turn to the MCP Server Log after enabling any MCP surface (the Copilot agent, the MCP shim, or direct on-prem MCP) to monitor and audit programmatic report access: to confirm which report a Copilot query retrieved, to review who was turned away and why after tightening an access rule, or to prove for a compliance file that a report was emailed to a specific recipient.

What each event records

Every event is a summary row backed by a full drill-in.

Captured fact	Detail
Time	When the event occurred.
Outcome	Success, Tool error, Not found, Access denied (the tool ran but the caller was not entitled to the resource), or Gate denied (rejected before any tool ran).
Tool	The MCP tool called, for example list reports, fetch an output, fetch a report's markdown, or email an output.
Target report	The resolved report display name and full folder path, falling back to the report id.
Workspace	The workspace id and name the call resolved to.
Caller	The user id, email, and name under the OAuth identity path, or "System" for the API-key and relay path.
Client	Derived from the client's User-Agent, for example "Sydney" for Copilot, or "claude-ai" / "claude-code" for Anthropic clients.
Transport	The ingress, "direct" or "cloudhub-relay". See Copilot, MCP, and trust paths.
Duration	How long the call took.
Trace and session	A trace identifier and a derived session key (persisted, not surfaced in this view).

Credentials are never written to a record, the bearer token or API key is not stored. The drill-in shows a Call card, a Caller card, an error callout when the call failed, and the full **Request** and **Result** panels with Formatted and Raw toggles and a per-body download. A markdown result renders as markdown. The request arguments and result body are stored in full, so the record holds the complete exchange.

Outcomes and denials

Outcome	Meaning
Success	The tool ran and returned a result.
Tool error	The tool ran but failed.
Not found	The requested resource was not found, or was trimmed away from the caller.
Access denied	The tool ran but the caller was not allowed the resource.
Gate denied	The request was rejected at the authentication gate before any tool ran. The specific reason rides in the deny reason, including `DirectChannelDisabled` and `CloudHubChannelDisabled` for the two ingress channels.

The access decision behind each Access-denied or Not-found outcome is enforced upstream by the content service's ACL and secured-list trimming, not by the log. A caller who is not entitled to a report gets an Access-denied or Not-found row, never the report's data.

Toolbar and controls

Control	Values / default
Time range	Last hour · Last 24 hours (default) · Last 7 days · Last 30 days · This month · All time.
Outcome filter	All outcomes (default), or one of Success, Access denied, Not found, Tool error, Gate denied. Combines with the tool filter.
Tool filter	All tools (default), or a specific tool.
Search	Free text, matching user, client, tool, report, or error.
Columns	Toggle Time, Outcome, Tool, Target report, Workspace, User, Client, Transport, Duration.
Refresh	Rebinds the grid.
Export to Excel	Downloads the log as an Excel workbook.
Page size	100 rows per page, paged server-side so the whole log is never loaded into memory.

Enablement and retention

Setting	Default	Notes
Enable MCP audit logging	On	Set at **Settings ▸ Configuration ▸ Audit Logging**. When off, new tool calls are not recorded but already-captured events still display, with a callout.
Retention window	90 days	Older records are purged automatically.

Auditing never breaks the tool call it observes: a record or query failure is swallowed and logged rather than surfaced to the caller.

Notes and limits

- **It audits the tool exchange, not the chat.** The AI model's conversation is the separate AI Interactions log.

- **The view is a flat activity list.** Session and thread grouping is persisted internally but not surfaced in this version.
- **Turning logging off stops new records only.** Existing records remain.
- **The Tool filter dropdown lists the original tool set.** The newer preview tool is audited but does not yet appear in the dropdown; use free-text search to find its rows until the list is refreshed.

> **Going deeper.** The MCP Server Log sits alongside the System, Job, AI Interactions, and Contribution logs in Audit logs. The Transport value it records is explained in Copilot, MCP, and trust paths.

Deploy the Office add-in

ADDIN-DEPLOYMENT | how-to | administrator | integration user

Report authors cannot use Reportworq's Excel and PowerPoint authoring until the companion add-in is installed. This page is for the **administrator** who deploys it to users who cannot or should not install it themselves, by Microsoft 365 Centralized Deployment or by a shared-folder sideload. For the add-in itself and what it does, see The Excel add-in.

When to use each method. Users with normal permissions can self-install from the Office Store (**Home ▸ Add-ins**, search "Reportworq"), and then admin deployment is not needed. Admin deployment is **required**, not just convenient, for:

- **Excel 2016 users**, who cannot self-install from the Store.
- **Microsoft 365 users without internet access**, who cannot reach the Store.
- **Microsoft 365 users without add-in-install permission**, whose policy blocks self-install.

Choose **Centralized Deployment** for those users when the Store path is available to you, and the **shared-folder sideload** as the on-prem alternative when the Store or central path is not usable.

Before you begin

- **SSL on the Reportworq server is a hard prerequisite.** Without HTTPS the add-in does not function, whatever the delivery method.
- A **licensed Reportworq account** is required on every use, not just at install.
- The add-in reaches the server at **port 8600** by default; a user can edit the server URL and port at sign-in.
- For Centralized Deployment, you have **Microsoft 365 tenant-admin** access to the admin center.
- For sideload, you have the manifest from Reportworq Customer Support (it is not shipped in the product) and a network share.

Deploy with Microsoft 365 Centralized Deployment

This is Microsoft-hosted and largely independent of the users' Office version.

1. In the Microsoft 365 admin center, go to **Settings ▸ Integrated apps**.
2. Choose **Add-ins ▸ Deploy Add-In ▸ Next ▸ Choose from the Store**.
3. Search for **Reportworq** and choose **Add**.
4. Accept the license and privacy terms.

5. Assign the recipient users or groups and choose **Deploy**.
6. Optionally validate the deployment, then choose **Next**, review Microsoft's announcement guidance, and **Close**.

Staged rollout. Deploy to a single test user first, validate against the Reportworq server, then return to the same procedure to add more users.

Sideload with a shared-folder Trusted Add-in Catalog

Use this when the Store or central path is not available, for example an air-gapped or Store-blocked estate. It configures each user's Excel Trust Center to load the add-in from a network share.

1. Obtain the **manifest** (`manifest-1.zip`) from **Reportworq Customer Support**. Sideload requires **Reportworq v5.0.0.82 or newer**; confirm the current minimum build with Support.
2. Place the manifest in a **shared network folder** (for example via **Properties ▸ Sharing ▸ Advanced Sharing**) and record the **UNC path**, for example `\\machinename\Add-in`. **All Reportworq users need read access** to the share.
3. In Excel, go to **File ▸ Options ▸ Trust Center ▸ Trust Center Settings ▸ Trusted Add-in Catalogs**.
4. Paste the UNC path in **Catalog URL**, choose **Add catalog**, tick **Show in Menu**, and confirm with **OK** (twice).
5. **Close and reopen Excel.**
6. Go to **Home ▸ Add-ins ▸ More Add-ins ▸ Shared Folder** and **Add** the Reportworq add-in.

The add-in installs into both Excel and PowerPoint.

Upgrade a sideloaded manifest

Replacing the manifest on the share is not enough on its own, the old one stays loaded until you clear the cache:

1. On the **Trusted Add-in Catalogs** screen, **clear the add-ins cache**.
2. **Close Excel.**
3. Replace the manifest file in the share with the newer version.
4. Restart Excel, and reinstall the add-in if prompted.

Notes and limits

- **SSL is mandatory.** The add-in will not function against a non-SSL server, regardless of delivery method.

- **A licensed account is needed on every use**, not only at install.
- **Sideload is per-machine.** Each user's Trust Center must be configured, and every user needs read access to the share.
- **Skipping the cache-clear leaves the stale manifest active** after a manifest upgrade.
- **Centralized Deployment requires tenant-admin rights.**

> **Going deeper.** For the add-in's companion pane, list-report builder, published-view import, and chart/map tools, see The Excel add-in. For the Microsoft 365 registration behind the Microsoft-hosted flows and Entra sign-in, see Microsoft 365 OAuth setup.

Connect SQL, OLE DB, and ODBC

how-to administrator

Reportworq reads from relational databases through three built-in connectors: **SQL Server**, generic **OLE DB**, and generic **ODBC**. All three expose tables, views, schemas, and columns, and back either a table pick or a raw SQL query in a data model. This guide covers configuring the connection, keeping credentials out of the connection string, and the driver and timeout traps that cause most relational failures.

Before you begin

- You are an administrator, working in **Settings** > **Integrations** > **DATASOURCES**. See Connect a data source for the general add-and-test flow.
- The database is reachable from the Reportworq server.
- The driver or provider is installed on the Reportworq server, not just your workstation. For ODBC this means a 64-bit driver; see the DSN traps below.
- You have credentials, or for Windows and trusted authentication, the server's service account has the required database permissions.

Configure a SQL Server connection

1. Add the **SQL Server** connector and open its editor.
2. Enter the server, database, and credentials.
3. Optionally set a **Validation Query**, the query that **Test connection** runs to verify the connection.
4. Save and select **Test connection**. A valid, reachable database returns **"Success."**

Configure an OLE DB or ODBC connection

OLE DB and ODBC take a freeform **Connection String** rather than discrete server and database fields.

1. Add the **OLE DB** or **ODBC** connector and open its editor.
2. Enter the **Connection String** for your provider or driver.
3. Keep credentials out of the string using the substitution variables below.
4. Optionally set a **Validation Query** for the connection test.
5. Save and select **Test connection**.

Keep credentials out of the connection string

Do not put a username and password directly in the connection string. Instead, put the placeholders `%USERNAME%` and `%PASSWORD%` in the string and enter the real values in the

separate masked **Username** and **Password** fields. Reportworq substitutes them at connect time, so the secrets never appear in the visible or stored string, and they are stored encrypted. Copy-links in the editor insert each token for you.

Connection-string examples

SQL Server takes discrete server, database, and credential fields rather than a connection string. OLE DB and ODBC take a freeform string that depends on the provider or driver installed on the Reportworq server. The examples below use the credential placeholders; substitute the driver or provider name for the one actually installed on your server.

- **SQL Server via OLE DB** (Microsoft OLE DB Driver): `Provider=MSOLEDBSQL;Data Source=DBSERVER;Initial Catalog=Finance;User ID=%USERNAME%;Password=%PASSWORD%;`
- **SQL Server via ODBC**: `Driver={ODBC Driver 18 for SQL Server};Server=DBSERVER;Database=Finance;UID=%USERNAME%;PWD=%PASSWORD%;`
- **PostgreSQL via ODBC**: `Driver={PostgreSQL Unicode(x64)};Server=dbhost;Port=5432;Database=finance;Uid=%USERNAME%;Pwd=%PASSWORD%;`
- **MySQL via ODBC**: `Driver={MySQL ODBC 8.0 Unicode Driver};Server=dbhost;Database=finance;User=%USERNAME%;Password=%PASSWORD%;`
- **CSV or flat file via the Microsoft Text Driver** (OLE DB or ODBC): point the driver at the folder holding the file, then read it with the `=RWSQL` function so the report re-pulls the file on every run instead of a manual copy and paste.

Configure a SQLite connection

SQLite is a built-in relational connector for a file-backed SQLite database, on the same base as the SQL connectors. It appears in the catalog as **SQLite**.

1. Add the **SQLite** connector and open its editor.
2. Enter the **Connection String**, normally just a file path in the form `Data Source=<path>`, for example `Data Source=C:\Reportworq\data\finance.sqlite`.
3. If the file needs credentials, use the same `%USERNAME%` and `%PASSWORD%` placeholders and masked fields as OLE DB and ODBC.
4. Save and select **Test connection**.

The file path resolves on the Reportworq **server**, not on your workstation, so the database file must be readable by the server's service account. SQLite defaults its row-cap dialect to **LIMIT**. For a shared, multi-writer database, use a server database instead of a SQLite file.

Advanced Options: timeouts

The collapsible **Advanced Options** section carries two timeouts:

- **Command Timeout**, in seconds. A value of **0 means wait indefinitely**. Set a non-zero value on any source where a runaway query should be bounded.
- **Connection Timeout**, in seconds.

There is also a connection-level **Limit Query Type** dropdown that selects the row-cap dialect the connector emits: **SELECT TOP** for SQL Server, **LIMIT** for MySQL or PostgreSQL, **FETCH FIRST n ROWS ONLY** for Oracle or DB2, or **Unavailable** when the source has no row-cap clause.

Query the source in a data model

Once the connection tests, a model author builds views over it on the model's Connection tab, choosing a **Query type**:

- **Table or View** picks a table or view from a filterable list. Table mode can select many tables at once to create several views in one action.
- **SQL Query** writes raw SQL in the editor, with a Format action. A SQL script can reference fields from any tables in the datasource, and supports `{{variable}}` double-brace tokens. Preview it before applying it to the model.

Text variables in SQL need single quotes: a `{{var}}` is substituted as a raw text replace with no automatic quoting, so a value that must become a SQL string literal has to be wrapped, for example `'{{state}}'` resolving to `'Maine'`. Numeric values are bare.

Connect CData sources

Several connectors reach their source through a **CData** driver. CData drivers wrap an API or SaaS system and present it to Reportworq as a relational, tabular source, so you query it with the same table or SQL modes as any database. Reportworq uses CData so that systems with no native SQL interface, such as Salesforce or Jira, can still feed a data model and be refreshed on every job run.

Supported CData connectors

Reportworq ships CData drivers for the sources below. **Each name links to CData's own ADO.NET documentation for that driver**, which is the authoritative reference for its connection properties, the tables it exposes, and the authentication options it supports. Go there whenever you need a property that is not on the Reportworq form.

Source	Driver documentation
Azure Analysis Services	ADO.NET provider docs
Databricks	ADO.NET provider docs
Dynamics 365	ADO.NET provider docs
Excel (a workbook exposed as a tabular source)	ADO.NET provider docs
HubSpot	ADO.NET provider docs
Jira	ADO.NET provider docs
NetSuite (can also import a saved-search schema)	ADO.NET provider docs
Power BI XMLA (the tile **PowerBI XMLA - SQL**, distinct from the native Power BI / SSAS ADOMD connector)	ADO.NET provider docs
SAP HANA	ADO.NET provider docs
Salesforce	ADO.NET provider docs
Smartsheet	ADO.NET provider docs

One source customers sometimes expect here is handled differently: **Snowflake** ships as a pre-configured **OLE DB** connector, not CData. See Configure an OLE DB or ODBC connection.

Configure a CData connection

1. Add the CData connector you need and open its editor.
2. Provide the connection details. Most drivers accept a **standard connection string** of CData connection properties, and most also support **OAuth** for authentication. For an OAuth driver, complete the interactive **Authenticate** step, or supply the service-principal or client credentials the driver requires, so Reportworq can obtain and refresh a token.

3. Where the driver exposes separate credential fields, enter secrets there rather than in the visible connection string.
4. Save and select **Test connection**.
5. Query the source in a data model through table or SQL mode, exactly as for the SQL connectors above.

For per-connector traps, such as the Salesforce security token or the Power BI XMLA workspace and Azure app registration, see the Connector catalog.

Find the connection properties for a driver

Each CData driver has its own connection properties and authentication options. To find them, start from the CData drivers documentation, open the driver for your source, then open its **ADO.NET** online documentation, which lists the connection-string properties and the OAuth setup for that source.

Notes and limits

- **ODBC DSN traps.** The ODBC driver must be installed on the Reportworq server. The DSN must be **64-bit** (a 32-bit DSN fails under the 64-bit service) and a **System DSN** (a User DSN can be invisible to the service account). For Windows or trusted authentication, the server's service account needs the database permissions. ODBC is the common path for most databases; OLE DB is mainly for Microsoft providers.
- **A very long SQL query in one Excel cell.** The relational query function is `=RWSQL` (internally `RW.Relational.Export`). If a SQL string is too long for one Excel cell, split it across several cells and concatenate them in the `=RWSQL` argument (for example `=$A$1&$A$2&$A$3`). Every fragment cell that is a formula must start with `=`, otherwise its leading quote is embedded literally into the string and the query silently fails. Validate the reassembled SQL in a database tool before deploying, and watch for missing table aliases once the fragments are joined.
- **CData sources build on this base.** Many API and SaaS systems (Salesforce, Jira, NetSuite, and more) are exposed as relational datasources through CData drivers on this same base. See Connect CData sources above and the Connector catalog. Note that **Snowflake** ships as OLE DB, not CData.

> **Going deeper.** To curate columns into governed business fields, set filters, and publish a view for report authors, see Data models.

The AI prompt store

reference | administrator | integration user

Reportworq's AI features, the AI Profile draft, chart auto-fill, the Ask AI assistant, and job diagnostics, each send an internal instruction prompt to your configured AI provider. The **AI prompt store** centralizes those prompts into a single global catalog you can review and tune in **Settings ▸ AI Prompts**, so you can adjust how Reportworq instructs the model without a code change or redeploy. Every prompt keeps a shipped default, so it can always be recovered.

Scope and access

Aspect	Value
Scope	System-scoped (global). One prompt set per Reportworq instance, not per workspace.
Access	Administrator only.
Default source	Code-embedded factory defaults, always available.
Editor	Settings ▸ AI Prompts.

How resolution works

- The catalog holds each prompt's factory-default text, its variable spec, name, description, and where it is used. That default is the source of truth.
- When you edit a prompt, Reportworq stores a per-prompt **override**. At render time the feature uses the override if one is present and non-empty, otherwise the shipped default.
- Resolution is defensive: if the stored copy is missing or unreadable, the feature logs the problem and falls back to the shipped default, so an AI feature never breaks because the store was edited or cleared. A wiped prompt folder simply means every prompt runs on its shipped default.

Variables

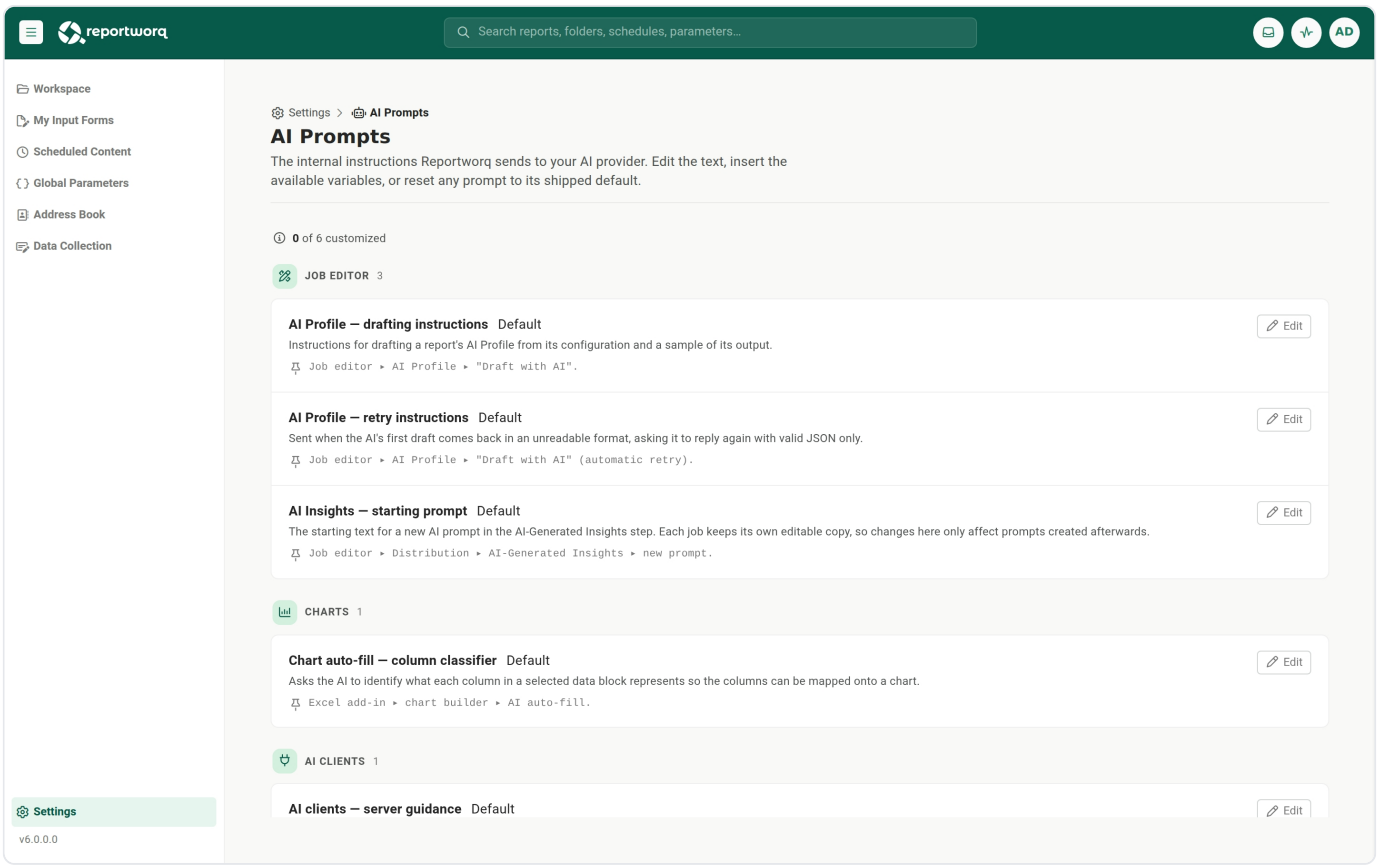
Prompts interpolate `{{name}}` tokens, using double braces so that any literal single-brace JSON inside a prompt body stays unambiguous. Each variable declares a name, a description, a required flag, and an optional example, and the editor renders them as an insertable palette.

- A **required** variable's token must remain in an edited prompt, or the save is blocked.
- An **unknown** token raises a non-blocking warning, it does not stop the save.
- A missing value substitutes as empty. Substitution is a single pass.

Factory reset

Choose **Reset to default** to clear the override, so the prompt reverts to the catalog text. Saving text that is identical to the default is treated the same as a reset. Opening the editor also materializes every shipped prompt into the store, refreshing the shipped metadata and default while preserving any override you have made.

Shipped prompts



Prompt id	Area	Variables	Where it is used
`job.ai-profile.system`	Job editor	none	AI Profile step, "Draft with AI"
`job.ai-profile.repair`	Job editor	`parseError` (required)	AI Profile draft, JSON repair round
`chart.ai-fill.classify`	Charts	`chartType` (required), `dataRows` (required)	Excel add-in chart builder AI auto-fill
`assistant.chat.system`	Assistant	`serverGuidance` (optional), `availableTools` (required)	In-app Ask AI chat assistant
`job.ai-diagnostics.system`	Diagnostics	none	Run-history AI diagnosis and the diagnostics loop
`job.ai-diagnostics.tool-catalog`	Diagnostics	none	Diagnostics AI first user message

When to use it

- Soften or reword the assistant's grounding instructions, or add an organization-specific instruction to the AI Profile draft prompt, to standardize how Reportworq prompts the model across the product.
- Tune the chart classifier prompt to your data conventions, then reset it when a Reportworq update ships an improved default.
- Recover a prompt that was edited into an unusable state with a single **Reset to default**.

Notes and limits

- Editing a prompt only changes the instruction text and its variables. Structural contracts that a parser must match (for example, JSON response schemas) stay in code and are not part of the store.
- A prompt is only actually sent when the underlying feature runs, which requires an AI provider connection. The store governs the text; the connection supplies the model.

> **Going deeper.** For the assistant that consumes `assistant.chat.system`, see The AI assistant.

The Script Runner

how-to | administrator | integration user | report author

The Script Runner lets you attach `.csx` C# script files to a distribution job and choose which points in the execution pipeline each one fires at. A script receives the live workbook or presentation, the job's parameters, and the job result, so it can validate, reshape, protect or report on the output **inside the run**, before anything is distributed.

It is the in-process alternative to a VBA macro or an external post-processing pipeline. What leaves the building is already correct, rather than fixed up afterwards.

> **Scripts run on the Reportworq server.** A script has access to local system resources through the account the Reportworq service runs as. Treat adding a script to a job the way you would treat any other server-side permission, and keep your `.csx` files somewhere with change control.

Licensing

The Script Runner is part of the **API** license capability, the same one that enables the REST API and webhooks. There is no separate Script Runner entitlement, and the License Details view lists it as *"API (includes Script Runner)"*.

If the license does not grant API:

- The **Custom Scripts (Advanced)** section does not appear in the job editor at all.
- A job that already carries scripts still **runs to completion**; each script is logged and skipped rather than failing the job. This is what you will see if a job is imported from a licensed install, or if a license lapses.

If you need the Script Runner, ask for **API** on your license.

The six hooks

Each hook blocks the pipeline: every script attached to it runs to completion before the job continues.

Hook	Fires	How often	Can reach
Before Processing	Before distributor pre-flight and any report calculation	Once per job step	Job metadata only
Before Calculation	Before a report's data-source calculation	**Once per report**	The workbook
After Calculation	After a report's data-source calculation	**Once per report**	The workbook
After Packet Generation	After all reports are merged into the final workbook, before format conversion	Once per job step	The merged workbook
After PowerPoint	After the PowerPoint file is generated	Once per job step	The presentation
After Send	After every distributor has been called	Once per job step	The job result

Three things to hold on to:

- **The two calculation hooks fire once per report**, not once per job. A job with four reports runs them four times each.
- **After Packet Generation operates on the merged workbook**, so a change there reaches every output format derived from it: Excel, CSV, Markdown and PDF.
- **After Send fires whether distribution succeeded or failed**, and by then the files have already gone out. Nothing a script does at that point can recall them.

Add a script to a job

1. Open the distribution job and open the **Job Options** sidebar.
2. Scroll to **Custom Scripts (Advanced)**. If you do not see it, the license does not grant API.
3. Select **Add Script**.
4. On the new row:
    - **File**: select the filename, or use **Select Script...** in the row's **...** menu, to open the file browser and pick a `.csx` file. The picker shows all file types, because there is no `.csx` filter, so navigate to your file.
    - **Hooks**: use the multi-select to choose one or more hooks. A single script can fire on several, and it can tell which one it is in by reading `HookName`.
5. Repeat for further scripts.

Scripts fire in list order within each hook. Use **Move Up** and **Move Down** in the **...** menu to reorder. A row with no hooks selected never fires.

Validate before you run

Validate in the row's **...** menu compiles the script and reports either:

```
'filename.csx' compiled successfully.
```

or the list of compiler errors.

This checks syntax and types only. It does not execute the script, so a reference to a worksheet that does not exist, or a path that is wrong, still fails at run time. Validate catches typos, not logic.

Disable without deleting

Disable in the **...** menu skips a script at every hook while keeping its file and hook configuration. The row dims to show it is inactive. **Enable** brings it back. Use this rather than removing a row when you are isolating a problem.

Remove deletes the row outright.

Where scripts are stored

A `.csx` is loaded through a **source report provider**, not from a server path. It lives wherever your report sources live, a network folder, SharePoint, Git, and it inherits that provider's access control.

That makes the provider worth choosing deliberately: whoever can put a file there can have it run. See Source report providers.

Reading the job log

Every script execution writes to the job history log, prefixed with the hook and the row it came from:

```
[Script/AfterCalculation/#1 validate.csx] Executing script for job 'Monthly Sales', output
'Monthly Sales - East'.
[Script/AfterCalculation/#1 validate.csx] Script completed in 42 ms.
```

Anything the script logs itself, and any warning or error it raises, carries the same prefix. In a job with several scripts, the `#N filename` part is how you tell which row produced a line.

Script time also appears in a run's performance breakdown, so a slow hook is visible rather than mysterious.

What to watch out for

- **Hooks are blocking.** A script that makes a slow HTTP call extends the job by exactly that long. Always set a timeout.
- **Halting is immediate and total.** A script that calls `AddError` stops the whole job: no further scripts in that hook, no further hooks, no distribution.
- **Cancellation is honored.** Canceling a running job stops its scripts at their next `await` point.
- **NuGet packages cannot be added at run time.** You have every assembly already loaded by the Reportworq process, and nothing else.
- **Reordering is by menu, not by drag.** The section's help text mentions dragging rows; use **Move Up** and **Move Down**.

Related pages

- Script API reference for everything a script can call.
- Example scripts for 29 examples covering every hook, downloadable as a single zip. Check each one's compile status on that page before you copy it.
- Source report providers for where `.csx` files are loaded from.
- REST API overview for the other half of the API license capability.
- Licensing for what the API capability covers.

Connect Power BI and Analysis Services

how-to administrator

Reportworq connects to **Power BI** semantic models (datasets) and to **Azure** or **SQL Server Analysis Services** tabular models over the **XMLA endpoint**, using the native **ADOMD** client. In the connector catalog this tile is **PowerBI XMLA - ADOMD**, and once created the connection's type reads **Power BI / SSAS (ADOMD)**.

There are two different ways to reach a Power BI XMLA endpoint, and they are separate connectors:

Tile	Driver	Query languages
PowerBI XMLA - ADOMD	Native ADOMD client	**DAX**, **DMV**, and **MDX**
PowerBI XMLA - SQL	CData driver	**SQL** only

This page covers the **ADOMD** connection. For the CData one, see the Connector catalog. Both return their results through the same worksheet formula, `RW.Relational.Export`.

Before you begin

- You are an administrator. Only administrators can add or configure datasource connections.
- **For Power BI:** the workspace is on a **Premium**, **Premium Per User (PPU)**, or **Fabric** capacity. The XMLA endpoint is not available on shared or Pro capacity. Azure Analysis Services models are always XMLA addressable.
- **For Power BI:** the **XMLA endpoint** is set to **Read** (or **Read Write**) in the capacity or tenant settings.
- You know the **workspace name** and the **dataset (semantic model) name** you want to query.
- You have credentials for the authentication mode you plan to use. See Choose an authentication mode.

Add the connection

1. Go to **Settings** > **Integrations**, then select **Add integration**.
2. Find **PowerBI XMLA - ADOMD** in the catalog and select **Add**. The connection editor opens.
3. Select **Enable datasource connection**.
4. Give the connection a **Datasource Name**. This is the name your `RW.Relational.Export` formulas refer to, so it must be unique across the instance

and is awkward to change later.

5. Choose an **Authentication Mode** and complete its fields, as described below.
6. Enter the **Connection String**, as described in Connection strings.
7. Select **Test connection**, then **Save**.

Datasource settings

Setting	What it does
Enable datasource connection	The gate that makes the connection usable. A disabled datasource cannot be selected or refreshed.
Datasource Name	The friendly name referenced from `RW.Relational.Export`.
Authentication Mode	**Password Authentication**, **OAuth (Service Principal)**, or **OAuth (Interactive Login)**.
Connection String	The XMLA endpoint, and optionally the model. Supports the `%USERNAME%` and `%PASSWORD%` placeholders.
%USERNAME% Variable	The value substituted for `%USERNAME%`. Masked, and stored encrypted.
%PASSWORD% Variable	The value substituted for `%PASSWORD%`. Masked, and stored encrypted.
Validation Query	The query **Test connection** runs. Defaults to `SELECT * FROM $SYSTEM.DISCOVER_CATALOGS`, a lightweight DMV that works against both Power BI XMLA and Analysis Services.
Command Timeout	Under **Advanced Options**, which is collapsed by default. Query timeout in seconds, default `30`, range 1 to 3600.

Connection strings

The connection string identifies the **XMLA endpoint** (the `Data Source`) and, optionally, the model (`Initial Catalog`).

Power BI

```
Data Source=powerbi://api.powerbi.com/v1.0/myorg/MyWorkspace;Initial Catalog=MyDataset
```

`MyWorkspace` is the Power BI **workspace name**, and `Initial Catalog` is the **dataset or semantic model name**.

Azure Analysis Services

```
Data Source=asazure://<region>.asazure.windows.net/<servername>;Initial Catalog=<modelname>
```

On-premises SQL Server Analysis Services (tabular)

```
Data Source=<server>\<instance>;Initial Catalog=<database>
```

Keep credentials out of the connection string

Anywhere in the connection string you can write the tokens `%USERNAME%` and `%PASSWORD%`. When the connection opens, Reportworq replaces them with the values from the **%USERNAME% Variable** and **%PASSWORD% Variable** fields, both of which are masked in the editor and stored encrypted. This keeps the real credentials out of the visible connection string. The editor's field labels are clickable and copy the token to your clipboard.

Choose an authentication mode

OAuth (Service Principal)

Use this for **unattended, server-to-server** access. It is the right default for scheduled reporting. Reportworq acquires an Entra (Azure AD) access token using the **OAuth 2.0 client credentials** grant and presents it to the XMLA endpoint. No user is involved, no credentials appear in the connection string, and the token is refreshed automatically before it expires.

Select **OAuth (Service Principal)** and provide:

Setting	Value
Tenant ID	The Entra (Azure AD) tenant, or directory, ID.
Client ID (Application ID)	The app registration's Application (client) ID.
Client Secret	The app registration's client secret. Masked, and stored encrypted.
OAuth Scope (optional)	Leave blank to use the default Analysis Services scope, `https://analysis.windows.net/powerbi/api/.default`. Override only if your environment requires a different scope.

With this mode the **Connection String** contains only the endpoint, with no user or password:

```
Data Source=powerbi://api.powerbi.com/v1.0/myorg/MyWorkspace;Initial Catalog=MyDataset
```

Two things must be true before it will work:

- The app registration is granted access to the target Power BI **workspace**.
- In the Power BI **tenant admin settings**, *Allow service principals to use Power BI APIs* is enabled, and the **XMLA endpoint** is set to **Read** (or **Read Write**).

OAuth (Interactive Login)

Use this when the model should be read **as a named person** and that person can sign in once. Reportworq uses the OAuth authorization-code grant: you sign in through the browser, and the resulting access and refresh tokens are stored on the connection and refreshed automatically at run time.

Select **OAuth (Interactive Login)** and provide the same **Tenant ID**, **Client ID**, **Client Secret**, and optional **OAuth Scope** as the service-principal mode. Two extra fields appear:

- **Redirect URI**, which is read only. Register this exact URI, of type **Web**, on the Entra app registration, and grant the app delegated permissions for your Analysis Services or Power BI resource.
- **Sign In**, with **Authenticate...** and **Sign out...** actions and a status line reading either **Signed in** or **Not signed in**.

To sign in:

1. **Save your changes first.** Selecting **Authenticate...** with unsaved edits returns *"Please save changes before performing this operation."* The same applies to **Sign out...**.
2. Select **Authenticate...**. Sign-in opens in a new browser tab, and the status updates to **Signed in** when it completes.

> **The browser address must be HTTPS, or `localhost`.** Interactive sign-in cannot complete over plain `http://` on a non-localhost address, and Reportworq stops with a message saying so.

Password Authentication

Use this to connect **as a specific organizational user** without an app registration. The credentials travel in the connection string, so use the `%USERNAME%` and `%PASSWORD%` placeholders and keep the real values in the encrypted variable fields.

Select **Password Authentication** and set the **Connection String** with the placeholders:

```
Data Source=powerbi://api.powerbi.com/v1.0/myorg/MyWorkspace;Initial Catalog=MyDataset;User ID=%USERNAME%;Password=%PASSWORD%
```

Then fill in **%USERNAME% Variable** (for example `analyst@contoso.com`) and **%PASSWORD% Variable**.

> **Accounts that require interactive multi-factor authentication cannot use this mode.** Report generation runs unattended on the server, so "user login" here means organizational account credentials supplied through the encrypted variables, not a browser prompt. If the account is subject to MFA, use **OAuth (Service Principal)** for unattended access, or **OAuth (Interactive Login)** if a person can sign in once.

Query the model

Reference the connection by its **Datasource Name** from the `RW.Relational.Export` worksheet formula. The formula runs the query text you give it and imports the result into an Excel range or table. The ADOMD connection accepts **DAX**, **DMV**, and **MDX**.

DAX

```
=RW.Relational.Export(A1, "PowerBI ADOMD", "EVALUATE SUMMARIZECOLUMNS('Date'[Year], ""Revenue"", [Total Revenue])")
```

DMV, for metadata discovery

```
=RW.Relational.Export(A1, "PowerBI ADOMD", "SELECT * FROM $SYSTEM.DISCOVER_CATALOGS")
```

MDX

```
=RW.Relational.Export(A1, "PowerBI ADOMD", "SELECT NON EMPTY [Date].[Year].Children ON ROWS, {[Measures].[Total Revenue]} ON COLUMNS FROM [Model]")
```

> **An MDX query must return a flat, tabular result.** If the result cannot be flattened into rows and columns, the formula reports an error. Rewrite the query so its result is tabular, or express it as DAX. For genuine matrix output, use the MDX query type's crosstab render mode in a data model rather than `RW.Relational.Export`.

Behavior and limits

- **Read only.** This connection queries models. It does not support write-back.
- **Tokens refresh automatically** in both OAuth modes. If a query fails because a token expired, Reportworq acquires a new token and retries once.
- **Long queries are bounded** by **Command Timeout**, which defaults to 30 seconds. Raise it under **Advanced Options** for expensive models.
- **Credentials are encrypted at rest**: the connection string, the `%USERNAME%` and `%PASSWORD%` variables, and the OAuth Tenant ID, Client ID, and Client Secret.
- **Test connection runs the Validation Query** and reports the row count. It confirms connectivity and authentication, but not per-query permissions inside the model.

Troubleshooting

Symptom	Likely cause	What to do
Test connection fails with an authorization error	Bad credentials, or the principal has no workspace access	For OAuth, verify the Tenant ID, Client ID, and secret, and that the service principal is a member of the workspace. For password mode, verify the account and password.
OAuth fails saying service principals are not allowed	The tenant admin setting is disabled	Enable *Allow service principals to use Power BI APIs*, and set the **XMLA endpoint** to Read or Read Write in the Power BI admin portal.
Cannot connect to a Power BI dataset at all	The workspace is not on Premium, PPU, or Fabric capacity	The XMLA endpoint requires one of those capacities. Shared and Pro workspaces have no XMLA endpoint.
A Connection String is required	The Connection String field is blank	Provide the `Data Source=...` endpoint string.
Tenant ID / Client ID / Client Secret is required	An OAuth mode is selected but a field is blank	Fill in all three, or switch to Password Authentication.
Please save changes before performing this operation	You selected **Authenticate...** or **Sign out...** with unsaved edits	Save the connection, then try again.
Interactive sign-in will not start	The browser address is plain `http://` on a non-localhost host	Reach Reportworq over HTTPS, or use `localhost`, then sign in again.
Password mode fails with an MFA or interactive sign-in error	The account requires interactive MFA	Use **OAuth (Service Principal)**, or an account exempt from interactive MFA.
An MDX query returns an error instead of data	The MDX result is not tabular	Rewrite the MDX to return a flat result, or express the query as DAX.

Related

- Connector catalog for the full connector list and the per-connector traps.
- Connect a data source for the general add, test, and manage flow.
- Connect SQL, OLE DB, and ODBC for the relational connectors that share the `RW.Relational.Export` formula.

- Relational database functions for the full `RW.Relational.Export` signature.

Script API reference

reference integration user

This is the reference for what a Reportworq `.csx` script can reach. For how to attach one to a job, see The Script Runner.

Script file format

Scripts are **plain C# statements**. No class, namespace or `Main` method is required, top-level code runs directly, and `await` is fully supported.

```
// Minimal example. No using statements required for the pre-imported namespaces.
var sheet = Workbook.Worksheets["Summary"];
sheet.Cells["A1"].PutValue("Generated by Reportworq");
sheet.Cells["A2"].PutValue(DateTime.Now.ToString("yyyy-MM-dd HH:mm"));
sheet.AutoFitColumns();
Logger.LogInformation("Summary stamped for {OutputName}.", OutputName);
```

Compilation environment

These namespaces are **pre-imported**, so you do not need `using` statements for them:

- `System`
- `System.Linq`
- `System.Collections.Generic`
- `Microsoft.Extensions.Logging`
- `Aspose.Cells`
- `Aspose.Slides`

Anything else needs an explicit `using` at the top of the script, for example `System.IO` or `System.Net.Http`.

These assemblies are explicitly referenced: `Aspose.Cells`, `Aspose.Slides`, `Microsoft.Extensions.Logging`, the .NET base library, and `System.Linq`. **Every other assembly already loaded by the Reportworq process is also reachable** with a `using`.

NuGet packages cannot be added at run time.

Globals available in every script

Name	Type	Notes
`JobName`	`string`	Name of the distribution job
`OutputName`	`string`	Tokenised output file name for the current runtime step
`Parameters`	`IReadOnlyDictionary<string, string>`	Resolved job parameter values
`HookName`	`string`	The hook currently firing, for example `"AfterCalculation"`
`EntryName`	`string`	This script's label, for example `"#1 validate_totals.csx"`
`Logger`	`ILogger`	Routes into the job history log

Hook-dependent objects

Name	Type	Non-null in
`Workbook`	`Aspose.Cells.Workbook`	Before Calculation, After Calculation, After Packet Generation
`Presentation`	`Aspose.Slides.Presentation`	After PowerPoint
`JobResult`	`ScriptJobResult`	After Send

All three are `null` in every other hook. **A script attached to more than one hook must check before use:**

```
if (Workbook == null)
{
    AddWarning("No workbook at this hook, skipping.");
    return;
}
```

To branch on the current hook, read `HookName`:

```
if (HookName == "AfterCalculation")
{
    // workbook work
}
else if (HookName == "AfterSend")
{
    // check JobResult
}
```

Logging and control flow

Member	Effect
`Logger.LogInformation` / `LogWarning` / `LogError`	Writes to the job history log with the `[Script/Hook/#N file.csx]` prefix
`AddWarning(string message)`	Logs a warning. **Execution continues.**
`AddError(string message)`	Logs an error and **immediately halts the job.** No further scripts and no further pipeline steps run.

`AddError` is the mechanism for script-controlled validation that must stop the pipeline. It throws, so nothing after it in your script runs either.

Calling `AddError` in After Send is valid but cannot undo a send. The files have already gone out. It records the failure in the job log; it does not recall anything.

Shared state across hooks

One `JobState` dictionary is allocated per job and threaded into **every script at every hook**. A value written in `Before Calculation` on report 2 is readable in `After Send`.

Method	Signature	Behavior
`SetValue`	`void SetValue(string key, object value)`	Stores a value. A null or empty key is silently ignored. A null value is stored but reads as absent.
`GetValue<T>`	`T GetValue<T>(string key, T defaultValue)`	Returns the value if present and assignable to `T`, otherwise the default. **Strict type match, no coercion**: store an `int`, ask for a `long`, get the default. Never throws.
`HasValue`	`bool HasValue(string key)`	True only for a stored non-null value.
`RemoveValue`	`void RemoveValue(string key)`	Silent no-op when the key is missing.

```
// In a Before Calculation script:
SetValue("report_start", DateTime.UtcNow);

// In an After Send script, a different row in the same job:
var started = GetValue<DateTime>("report_start", DateTime.MinValue);
Logger.LogInformation("Job took {Elapsed}.", DateTime.UtcNow - started);
```

Thread-safe file append

`AppendLineWithRetry` writes one line to a file, tolerating transient `IOException`. Use it instead of `File.AppendAllLines` whenever concurrent burst job steps might write to the same file.

```
void AppendLineWithRetry(
    string path,
    string line,
    int maxAttempts = 8,
    int initialDelayMs = 25)
```

It retries with jittered exponential backoff, per-attempt delays start at 25 ms and double, capped at one second each. It opens the file with read sharing so a tailing reader does not block it. The final attempt lets the exception escape, so a genuinely stuck file still surfaces as an error.

```
// After Send, safe for concurrent burst steps writing to a shared CSV
var line = $"{DateTime.UtcNow:O},{JobName},{OutputName},{JobResult.Success}";
AppendLineWithRetry(@"\\server\logs\reportworq_audit.csv", line);
```

The After Send job result

`JobResult` is populated only in After Send. All properties are read-only.

Property	Type	Meaning
`Success`	`bool`	Every distribution step completed without error
`HasCalcWarnings`	`bool`	Any report sheet had calculation warnings
`OutputFiles`	`IReadOnlyList<string>`	Names of all output files prepared for distribution
`Errors`	`IReadOnlyList<string>`	Errors collected during the job, empty on success
`StartTime`	`DateTime?`	UTC job start
`EndTime`	`DateTime?`	UTC job end
`Elapsed`	`TimeSpan`	Total elapsed time
`Status`	`string`	For example `Complete`, `Failed`, `Cancelled`
`StepName`	`string`	Runtime step name. Includes the contact name on a burst job.

```
if (!JobResult.Success)
{
    var errors = string.Join("; ", JobResult.Errors);
    Logger.LogWarning("Job failed in {Elapsed:g}: {Errors}", JobResult.Elapsed, errors);
}
else
{
    Logger.LogInformation("Job succeeded in {Elapsed:g}. Files: {Files}",
        JobResult.Elapsed, string.Join(", ", JobResult.OutputFiles));
}
```

Error handling

Scenario	Behavior
`AddError` called in any hook	Job halts immediately, remaining scripts and pipeline steps skipped, error recorded in job history
`AddError` called in After Send	Same, but the sends already happened
Unhandled runtime exception	Wrapped and propagated, halting the job
Compilation error	Thrown before execution begins, job halts
Job canceled	Cancellation propagates, job marked canceled, scripts stop at their next `await`
Any other exception in After Send	Caught and logged, job result recording continues

Script C# is not quite file C#

A `.csx` is compiled as a **script**, not as a source file, and the grammar differs in one place that bites immediately.

You cannot use a top-level `using` declaration. At the start of a statement, the script parser reads `using` as a using *directive*, so the C# 8 form fails to compile:

```
using var http = new HttpClient();   // does NOT compile in a .csx
```

The parser binds `using var` as a directive importing a namespace called `var`, then expects `;` and finds the variable name instead, reporting `CS1002` at the identifier. **Adding a semicolon does not help**, because the semicolon is not what is missing.

Write it as a using *statement*, or drop the `using` entirely:

```
using (var http = new HttpClient())   // compiles
{
    // ...
}

var http = new HttpClient();          // also compiles
```

Using *directives* at the top of the file are fine, and are how you reach namespaces that are not pre-imported. It is only the inline `using var x = ...` declaration form that the

script grammar rejects.

`Cell` is ambiguous

Both `Aspose.Cells` and `Aspose.Slides` are pre-imported, and **each defines a type called `Cell`**. Using the bare name fails to compile:

```
foreach (Cell cell in sheet.Cells)             // does NOT compile: CS0104, ambiguous
foreach (Aspose.Cells.Cell cell in sheet.Cells) // compiles
```

Qualify it. The same applies to any other type name the two libraries share.

`Cells` is a non-generic collection

`Worksheet.Cells` implements the non-generic `IEnumerable`, so `var` in a `foreach` over it binds as `object` and members like `.IsFormula` and `.Value` will not resolve. State the element type explicitly, qualified as above.

HTTP calls

`System.Net.Http.HttpClient` is available. **The hook blocks until the call completes**, so always set a timeout.

```
using System.Net.Http;

using (var http = new HttpClient { Timeout = TimeSpan.FromSeconds(10) })
{
    var response = await http.PostAsync(url, content);
}
```

Note the `using (...)` statement form rather than `using var`, for the reason above.

Log format

Each execution is bracketed by the runner:

```
[Script/AfterCalculation/#1 validate.csx] Executing script for job 'Monthly Sales', output
'Monthly Sales - East'.
[Script/AfterCalculation/#1 validate.csx] Script completed in 42 ms.
```

`AddWarning` and `AddError` use the same prefix:

```
[Script/AfterCalculation/#1 validate.csx] Summary!B2 is zero or blank, job halted.
```

Entries appear in the job history log for that job step.

Related pages

- The Script Runner for configuration, licensing and the security posture.
- Example scripts for working code against every hook.

Data models

how-to administrator

A **data model** is a reusable, governed layer between a source system and the people who build reports. It binds a datasource to one or more curated **views**, and those views are what report authors import when they **build Excel Reports in the Excel add-in**. A model lets you rename and reshape source columns into clean business fields once, so every report reads from a stable vocabulary that survives changes to the underlying source schema.

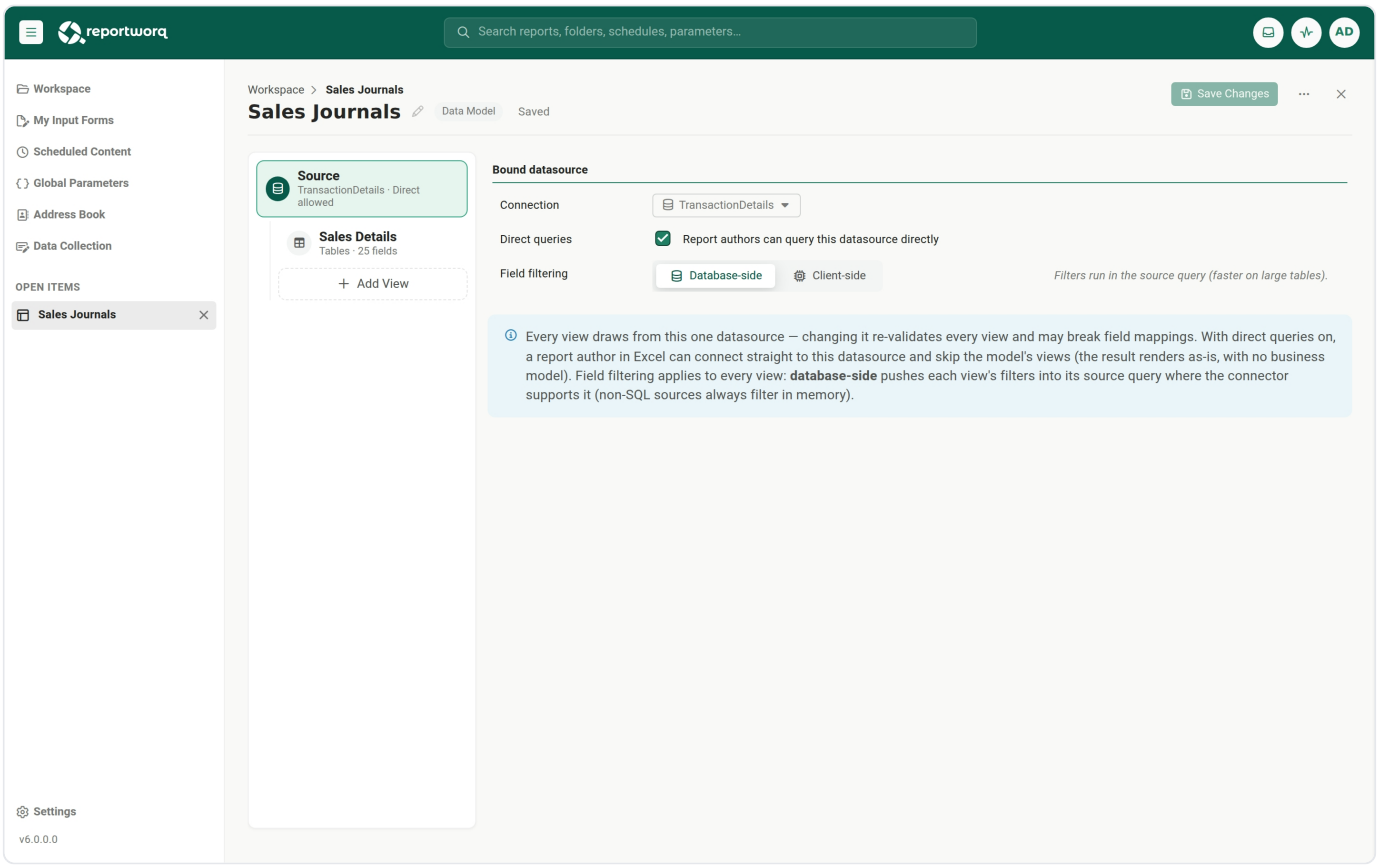
A model feeds two downstream workflows, not just reporting: **Distribution Jobs** (template-based output) and **Contribution Campaigns** (input forms). A contribution form binds to a model view the same way a report does, so curate a model as a shared vocabulary, not just a report source.

Before you begin

- At least one datasource connection exists and tests successfully. See Connect a data source.
- You have permission to author data models in the workspace.

The model editor at a glance

A model is edited in the workspace as one or more views. Creating a model is a two-step exercise: you create it against a datasource, then you edit it to curate fields, set filters and variables, and publish. The editor exposes a model-level **Source** tab and, per view, four sub-tabs.



- **Source tab (model level).** Binds the model to a configured datasource and carries two important model-wide settings: the **Direct queries** toggle and the field **filter mode**. Both are covered below.
- **Views.** A view is the model's unit of output: one source query plus a curated set of fields. **Add View** creates another view, and a model can hold several. Each view is built through its sub-tabs:
    - **Connection** defines the connector-specific source, for example a table for SQL or a cube data view for Planning Analytics. Use **Browse datasource** to explore tables and columns, then **Import Fields** (or **Update Fields** on a re-import) to generate the field list from the source.
    - **Fields** is the curated reporting field list, auto-generated on import. Each field decouples its reporting-facing **Id** and **Name** from the source column it maps to (the **Mapped Field**), so renaming or re-mapping a source column does not break downstream reports. Fields can be hidden, reordered, renamed, re-typed, filtered, added, or deleted.
    - **Variables** are model variables that parameterize the source query and flow downstream to reporting and bursting. A variable's default is what preview and reports resolve when nothing overrides it.
    - **Preview** runs the view and shows the resolved rows after variables and field projection are applied.

Curate the fields

The Fields tab is where a raw source becomes a governed model. Each field shows a green **mapped** chip or an amber **unmapped** chip, recomputed automatically as you edit.

- **Unmapped fields are excluded until you remap them.** A field that is unmapped, whether you added it without a mapping or it lost its mapping after a source change, is left out of the model, the preview, and every report until you point it back at a source column. Unmapped does not mean "author fills it in later", there is no calculated or literal field type; unmapped simply means excluded.
- **Field names must be unique within a view.** A duplicate display name is rejected and the edit reverts.
- **The Fields list is not paged.** Field order is set by dragging rows, and a drag cannot cross a page boundary, so every field is listed on one scrolling list however many there are. Other long lists in Reportworq page at 100 rows, this one deliberately does not.
- **Hidden fields stay queryable.** Hiding a field removes it from the report author's field list but keeps it usable as a filter reference, so you can filter on a key column without exposing it. Use **Hide all** or **Show all** to toggle visibility in bulk.
- **Quote a text value only when it lands in a SQL string literal.** A `{{variable}}` is substituted into the source query as a raw text replace, with no automatic quoting. When a variable feeds a **SQL query on a relational source** (SQL Server, OLE DB, ODBC, SQLite, or a CData source), a text value that must become a SQL string literal has to carry its own quotes in the query, for example `'{{state}}'` resolving to `'Maine'`; numeric values are bare. This is a SQL-syntax requirement of the query, not a property of the variable's default. Sources whose query text is not SQL, such as Planning Analytics and the planning-tool APIs, do not need the quoting.

Choose where filters run (filter mode)

The Source tab's filter mode decides where a model's field filters are evaluated.

- **Database-side (pushdown)** translates the filter into the source query, so the source engine returns only matching rows. Choose this for large sources and selective filters: it minimizes data transfer and leans on the source's indexes.
- **Client-side (in-memory)** retrieves the rows first and filters them inside Reportworq. Choose this when the result set is already small.

Important: pushdown is SQL-only. Non-SQL sources (Planning Analytics, the planning-tool APIs, and other non-relational connectors) always filter in memory regardless of this

setting. So for a TM1 or Workday Adaptive model, filters are applied client-side whatever the mode says.

Direct queries: modeled view or direct query

Direct queries is a Source-tab toggle, and it **defaults off**. Leaving it off keeps the model view-only: authors consume the curated, published views. Turning it on lets a report author query the datasource directly and bind to the query result's own columns, skipping the model's curated fields entirely.

When to use each:

- **Use a published view** for governed, reusable, change-resilient reporting. The field-mapping abstraction survives source schema changes, and the view is reusable across many reports and forms.
- **Use a direct query** for a one-off, well-understood, or exploratory pull, for example running a known SQL extract or inspecting a source before deciding whether to invest in a curated view. Direct-query results do not carry the field-mapping abstraction, so they do not survive source changes and are not centrally reusable.

Publish a view

Each view carries a **Draft** or **Published** state, tracked per view, so one model can hold a mix of both.

1. Build and preview the view in **Draft** while you are still shaping it. Draft views are hidden from report authors.
2. Switch the view to **Published** when it is ready. Only published views appear in the Excel add-in's **Connect** picker for report authors to import.

When to use the two states together: publishing gates the picker, not existing bindings. A report already bound to a view keeps resolving even after you pull that view back to **Draft**, so you can safely revise a published view in Draft while the live reports keep running, then re-publish once the revision is validated.

Notes and limits

- The Preview tab is a fixed, fresh 100-row run with no cache. Report display and formatting (grouping, aggregates, sort priority, format modes) are authored in the Excel add-in's list-report builder, not in the model editor.
- Re-importing an already-imported table creates an additional copy rather than updating the existing one. Edit the existing view in place instead. See the Connector catalog for the same trap with Pigment blocks.

- Opening a model in **View** mode redirects to **Edit** (for Authors) or **Properties** (for Users); a model has no separate read-only surface.

> **Going deeper.** For the field labels and query types of a specific source, see Connect IBM Planning Analytics, Connect SQL, OLE DB, and ODBC, and the Connector catalog.

Example scripts

reference integration user

Reportworq ships a library of `.csx` examples, one set per hook plus two that span every hook. They are the fastest way to see the script API used in context, and most are close enough to production use to adapt directly.

Every example is compile-checked against the real script API on each build, so the library cannot drift as the API changes.

Download

Download all examples (.zip)

The zip preserves the folder-per-hook layout below. Individual scripts are linked from each table.

Before you adapt one

- **Read the header comment.** Every example opens with its intended hook and a short explanation of what it does and what it assumes.
- **Check the hook matches.** An example written for `AfterCalculation` expects a workbook. Attaching it to `BeforeProcessing` gives it a null one.
- **Replace the placeholders.** Several carry constants you are meant to change: webhook URLs, file paths, passwords, sheet names.
- **Mind the script grammar.** A `.csx` is compiled as a script, not a source file, and two things catch people out: top-level `using var x = ...` declarations do not compile, and the bare type name `Cell` is ambiguous because both Aspose libraries are pre-imported. See Script C# is not quite file C#.
- **Validate after editing.** Use **Validate** on the script row to compile-check before you run the job.

Before Processing

Runs before distributor pre-flight and before any report is calculated. There is no workbook yet, so these are gate-keeping and preparation scripts.

Script	What it does
Block On Weekends	Block the job from running on Saturday or Sunday.
Copy ETL Export To Provider	Copy a fresh data extract from an ETL share into the report provider folder.
Validate Required File Exists	Halt the job if a required source file is missing.

Before Calculation

Runs once per report, before its data source is calculated. The workbook exists but is not yet populated, which is the moment to validate inputs and seed cells.

Script	What it does
Seed Settings Cells	Push job parameters into a "Settings" sheet before formulas run.
Set Calc Mode	Force deterministic formula calculation regardless of the source workbook's saved settings.
Stamp Metadata Sheet	Stamp a hidden "_Meta" sheet with generation context.
Validate Named Range Exists	Halt the job if a required named range is missing.
Validate Required Parameters	Halt the job if any required parameters are missing or blank.

After Calculation

Runs once per report, after its data source is calculated. The data is in the workbook, so this is where validation, formatting and redaction belong.

Script	What it does
Apply Currency Format	Apply a currency number-format to columns whose header mentions Amount, Total, or Revenue.
Redact PII Columns	Mask values in any column whose header looks like personally identifiable information (PII).
Sort Details Sheet By Column	Sort the "Details" sheet rows by column C, descending.
Stamp Watermark On All Sheets	Stamp every visible worksheet with a small watermark in cell A1.
Validate KPI Threshold	Halt the job if a KPI value is missing, negative, or absurdly large.
Validate Totals Not Zero	Halt the job if the Summary total cell is blank or zero after calculation.

After Packet Generation

Runs once, on the merged workbook, before format conversion. Changes here reach every output format derived from it, so this is the place to harden what gets distributed.

Script	What it does
Hide Staging Sheets	Hide any worksheet whose name starts with an underscore.
Protect Sheets With Password	Apply sheet-level protection across every visible worksheet.
Remove Empty Sheets	Remove worksheets that have no data after the packet was assembled.
Remove Formulas From Distribution Sheet	Replace every formula on the "Distribution" sheet with its calculated value.
Set Workbook Properties	Stamp the workbook's built-in document properties with job metadata.

After PowerPoint

Runs after the PowerPoint file is generated. The presentation is available; the workbook is not.

Script	What it does
Apply Footer And Slide Numbers	Brand every slide with a consistent footer and visible slide numbers.
Remove Draft Slides	Remove draft slides and any slide marked Hidden in the source deck.
Replace Tokens On Slides	Replace simple tokens in slide text without losing the existing formatting.
Stamp Slide Footer	Stamp every slide with a small footer text frame showing report name and generation date.
Validate Minimum Slide Count	Halt the job if the presentation has fewer than the expected number of slides.

After Send

Runs after every distributor has been called, whether distribution succeeded or failed. The job result is available, and the files have already gone out.

Script	What it does
Append Audit CSV	Append a one-line audit entry to a shared summary CSV.
Teams Webhook On Failure	Send a Microsoft Teams webhook notification when a job fails.
Trigger Downstream API	POST to a downstream HTTP API once distribution completes successfully.

Multi-hook examples

These attach to several hooks at once and branch on `HookName` to tell which one they are in.

Script	What it does
Audit Every Hook Event	Append a one-line CSV entry every time a hook fires during a job.
Random Failure Demo	A demo script that occasionally surfaces a comical error.

Related pages

- The Script Runner for configuration and licensing.
- Script API reference for every global, method and result property these examples use.