O M U M U

The Omumu Manual

Everything the platform can do

Edition 23 September 2026

3 of 15 chapters

The Omumu Manual
Edition 23 September 2026

Contents

CHAPTER 1

Selling

You have something worth paying for. This chapter is about the distance between that and money arriving: putting a price on it, giving buyers a checkout that works, and getting them into the product the moment they pay — all from your own site, on your own terms.

On Omumu you don't fill in a product listing on someone else's marketplace. You tell your AI assistant what you want to sell and for how much, and it sets up the offer, the checkout, and the delivery. You can pre-sell a course that doesn't exist yet, bolt an extra purchase onto your checkout, give a course away free to grow your list, invoice a company instead of asking for a card, or hand someone access as a gift. Every one of those is a few sentences of conversation. The rest of this chapter shows you the sentences — and, for each, the buttons in the admin UI that do the same thing.

Concepts

Selling on Omumu revolves around one object and one guarantee.

The object is the **offer**: a sellable bundle scoped to your site. An offer holds a list of products (courses, skills, and optionally a platform subscription plan), a price, a validity window, and an optional tag that triggers automations when someone buys. The offer — not the course — is what has a checkout URL. One

course can be sold through many offers at different prices and framings; an offer can bundle several products into one purchase.

The guarantee is that **an offer's price is frozen once it is published**. The moment an offer's validity window is set and it can be bought, its price never changes again. To charge something different, you copy the offer into a new one with a new price — the old offer id keeps meaning what every buyer saw when they bought. Your sales history stays honest by construction.

A few more terms carry the rest of the chapter:

A **bump offer** is an offer attached to a parent offer and presented as an optional checkbox on the order form — "add this to my order". Only bumps that are currently live are shown, and a ticked bump fulfils as its own mini-purchase alongside the parent. Since a bump must be live (hence published) to be offered, its price is already frozen too.

A **free offer** is simply an offer priced at zero. The order form detects the zero price and fulfils immediately — no payment step at all. Understand what that means: while a free offer is live, *anyone* who submits the public order form gets the products. It is a live giveaway, not a private test.

An **Invoice Request** is a payment option for buyers who don't pay by card at checkout — typically companies. Choosing it grants the buyer full access immediately, at the offer's real price, and records an invoice request for you to bill against later. Grant now, bill later. Unlike a free offer it sits behind an explicit button at a real price, so it is not open-door access.

Behind all of these sits **offer fulfillment**, and its key property is that it is **decoupled from payment**. Granting access and recording payment are separate machinery: the access side gives the buyer their student role, product access, welcome email, and fires the offer's automation tag; the payment side records the payment and sends the receipt. Card purchases run both; a free offer runs both with an amount of zero; an Invoice Request runs both without money changing hands yet; a gift grant (below) runs only the access side. Fulfillment is also exactly-once: however many payment signals arrive — buyer returning to the page, provider webhook, a retry — a purchase fulfils once, never twice.

Because fulfillment doesn't wait for a finished product either, Omumu supports the **pre-sale offer**: an offer connected to a real but still-empty course, sold to validate demand before you build. Buyers get normal access immediately — to a product you then fill in. The platform's **validation page** takes

this further: a designed, structured sales page sold through a low-ticket offer, built to test whether strangers will pay for an idea. This chapter covers the selling mechanics of pre-sales; the validation page itself and its self-improving machinery have their own chapters.

Hands-on

The walkthroughs below follow Mira, an independent developer turned educator who teaches non-programmers to automate their work with spreadsheets and AI. She runs her course business by talking to her AI assistant, which operates her Omumu site directly. Each walkthrough leads with the conversation, shows what happens underneath, and gives the admin-UI route.

Sell your course from your own website

Mira's flagship course "Automate Your Week" is ready. She tells her assistant:

> "Create an offer for Automate Your Week at 1 490 NOK, card and Vipps payments, and put it on sale from Monday."

The assistant creates the offer (`omumu_offer_create`) with the title, the price in cents, the course, the payment providers, and a `validFrom` date. Only a title and price are strictly required — currency defaults to NOK, and products, payment options, pages, and dates can all be added afterwards (`omumu_offer_add_product`, `omumu_offer_update`). The offer starts as *scheduled*, goes *live* when its window opens, and the create call returns the checkout URL Mira can put behind any buy button.

Mira asks "what am I selling right now?" and the assistant lists every offer with its status, products, and pricing (`omumu_offer_list`), or pulls one offer's full detail (`omumu_offer_get`). If a draft was a mistake, it can be deleted — but only an offer that has never been sold (`omumu_offer_delete`); sold offers are permanent sales history.

Two honesty notes. First, the frozen-price rule from Concepts applies here: once this offer is published, "change the price to 1 990" means the assistant copies it into a new offer at the new price — the update tool changes the price only while the offer is still unpublished. Second, whether the offer can

actually take orders also depends on your site's payment setup: card payments need your Stripe account connected and chargeable, Vipps needs your Vipps integration. Until at least one selected payment path is usable, the order form shows a neutral "not taking orders" state and un-pauses by itself when you finish the setup — no republish needed.

In the admin UI: the offer wizard at *Admin → Offers → New offer* walks through the same fields, and existing offers are listed and edited at *Admin → Offers*. There is also the Offer Lab, a guided interview that designs an offer and its sales page for you — that flow belongs to the Ad Validation and Offer DNA chapter.

Get paid for a course idea before you build it

Mira has an idea for a second course — "AI Email Triage" — and wants proof people will pay before she records a single lesson. She says:

> "Create an empty course called AI Email Triage, and sell it at an early-bird 790 NOK. I'll build it in March."

The assistant creates the skeleton course (chapter 2) and an offer around it, exactly as above. This is a pre-sale offer: a purchase fulfils like any other, so early buyers get real access immediately — to a course that is still empty. There is no special pre-order state and no gate; the "opens in March" date is a communicated expectation. Set it and the thank-you page tells buyers *"Available on [date]"* instead of showing an access button. As Mira fills the course in, the content simply appears for everyone who already bought.

Stated plainly: setting that access-opens date is **UI-only** today — it lives on the offer's edit page in the admin UI and is not exposed through the assistant's tools. The assistant can create and sell the pre-sale offer; Mira sets the date at *Admin → Offers → edit*.

If the idea fails the test, the exit is as honest as the entry: refund the buyers and revoke their access. And if what you want to validate is the *pitch* as much as the product, the validation page (see the Pages and Sales Pages chapter) is this same pre-sale mechanic wrapped in a designed, self-improving sales page.

Add an order bump to your checkout

Every buyer of "Automate Your Week" is a good prospect for Mira's template pack, sold as its own small offer at 290 NOK. Rather than a separate campaign, she puts it on the checkout itself:

> "Offer the template pack as an order bump on the Automate Your Week checkout."

The assistant attaches the bump (`omumu_offer_add_bump`). From then on the order form shows a checkbox — buy the course, tick to add the templates — and a ticked bump fulfils as its own mini-purchase alongside the parent: the buyer gets both products, and your records show both sales. Both offers must belong to the same site, an offer cannot be its own bump, and only a bump that is currently *live* is actually shown to buyers — so if the template-pack offer's validity window ends, the checkbox quietly disappears rather than selling something unbuyable. Removing a bump is the mirror move (`omumu_offer_remove_bump`).

In the admin UI: bumps are managed on the offer's edit page at *Admin → Offers*.

Run a free course as a giveaway

Mira made a three-lesson mini-course, "Five Formulas That Replace an Hour of Work", to grow her mailing list. She says:

> "Create a free offer for the mini-course."

An offer with a price of zero (`omumu_offer_create` with `totalAmountCents: 0`) skips the payment step entirely: anyone who submits the order form gets the course immediately, with the full delivery — account, student role, access, welcome email, and the offer's automation tag, which is what makes it a list-builder: tag the buyers and let your email automation (see the Audience and Email chapter) take it from there.

The honesty note is the one from Concepts, worth repeating because it bites: a live free offer is a live giveaway. The public order form fulfils it *for anyone who submits*. Do not use a free offer to preview a course privately — end the offer's validity window when the giveaway should stop, or use a direct access grant (below) for individuals.

In the admin UI: create an offer in the wizard at *Admin → Offers* with price 0.

Take invoice payments from companies

An office manager wants "Automate Your Week" for her team of five and asks — as B2B buyers do — whether she can pay by invoice instead of card. Mira tells her assistant:

> "Add invoice payment as an option on the Automate Your Week offer."

Invoice Request is a payment provider like card or Vipps, selectable when creating the offer (`omumu_offer_create` accepts it in `paymentProviders`). A buyer who clicks it gets full access immediately at the offer's real price, and Omumu records an invoice request for Mira to bill against. The trade is explicit: the buyer is served now, and collecting the money is Mira's follow-up, tracked so it isn't forgotten. Because it needs no card machinery, an offer with Invoice Request enabled can take orders even before Stripe or Vipps is connected.

Stated plainly: two parts of this flow are **UI-only** today. Toggling payment options on an *existing* offer happens on the offer's edit page (the assistant sets providers at creation time), and the follow-up — seeing and settling recorded invoice requests — is admin-UI work, not something the assistant does.

Give someone free access to a paid course

A podcast host who had Mira on as a guest should get "Automate Your Week" as a thank-you. No checkout, no zero-price offer visible to the world — just a direct grant:

> "Give jonas@example.com access to the Automate Your Week offer — note that it's the podcast thank-you."

The assistant grants access (`omumu_offer_grant_access`): Jonas gets everything a paying buyer gets on the access side — student role, access to every product in the offer, welcome email, the offer's automation tag — but no payment is recorded and no receipt is sent, because none exists. The optional message is kept as an audit note, so six months later "why does Jonas have access?" has an answer. Two constraints: the person must already exist as a user on your site, and offers containing a platform subscription plan can only be gift-granted by Omumu.

In the admin UI the equivalent is slightly different in shape: user management at *Admin → Users* grants access per *product* rather than per offer — pick the user, grant the course. Same outcome for a single course; the offer-level grant (all products in one move, offer tag fired) is the assistant's move.

Reference

`omumu_offer_add_bump`

Adds a bump offer to an existing offer. The bump appears as a checkbox on the checkout page. Both offers must belong to the same site. An offer cannot be its own bump.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`bumpOfferId`	string	yes	The bump offer ID to add
`offerId`	string	yes	The parent offer ID

`omumu_offer_add_product`

Adds a product (course or skill) to an existing offer. The offer must not be live or ended. Subscription plans (SAAS) are added via omumu_offer_add_subscription_plan instead.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID
`productId`	string	yes	The product ID to add — a course id or a skill id (the meta_product id, which equals the underlying course/skill id)

`omumu_offer_add_subscription_plan`

Adds a subscription plan (Omumu platform access) to an existing offer. When the offer is purchased, the buyer gets a new Omumu site provisioned on the plan's tier. Use SAAS_BASIC for the Omumu Basic plan.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID
`subscriptionPlanId`	string	yes	The subscription plan ID (e.g., SAAS_BASIC)

`omumu_offer_create`

Creates a purchasable offer with title, price, courses, and payment providers. The offer starts in 'scheduled' status. Use validFrom/validTo to control availability. Returns the offer with its checkout URL.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`buttonText`	string	no	Text for the purchase button (e.g., 'Buy now')
`courseIds`	array	no	Array of course ID strings to include in the offer
`currency`	string	no	Three-letter currency code (default: NOK)
`descriptionPageId`	string	no	Page ID for the offer description shown at checkout
`paymentProviders`	array	no	Array of payment provider names: STRIPE, VIPPS, INVOICE_REQUEST, NONE
`subscriptionPlanId`	string	no	Subscription plan ID to include (e.g., SAAS_BASIC for Omumu Basic). Bundles Omumu platform access into the offer.
`tagline`	string	no	A short tagline or subtitle
`thankYouPageId`	string	no	Page ID for the post-purchase thank-you page
`title`	string	yes	The offer title
`totalAmountCents`	number	yes	Total price in cents (e.g., 4900 for 49.00 NOK)
`validFrom`	string	no	Start date/time in ISO-8601 format (e.g., 2026-04-01T00:00:00)
`validTo`	string	no	End date/time in ISO-8601 format
`vatAmountCents`	number	no	VAT amount in cents (included in totalAmountCents)

`omumu_offer_delete`

Deletes an offer. Only offers that have not been sold can be deleted.

- **Write category:** standard write
- **Access:** requires the `OFFER_DELETE` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID to delete

`omumu_offer_get`

Gets an offer by ID with full details including checkout URL, products, pricing, and payment providers

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID

`omumu_offer_grant_access`

Grants a user access to all products in an offer WITHOUT requiring payment. This bypasses the normal checkout/payment flow entirely. The user receives the same access as if they had paid: STUDENT role, product access entries, welcome email, and offer-tag execution. An offer containing a SAAS plan can only be granted by Omumu. The user must already exist on the same site as the offer.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`email`	string	yes	Email of the user to grant access to. Must already exist on the same site as the offer
`message`	string	no	Optional free-text reason for the grant (e.g. 'Demo account for Ole-Arvid'). Recorded in the audit log only, not stored on the access entry. Max 500 characters
`offerId`	string	yes	The offer to grant access to

`omumu_offer_list`

Lists all offers for the current site with their status (scheduled/live/ended), products, and pricing

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

No parameters.

`omumu_offer_remove_bump`

Removes a bump offer from an existing offer.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`bumpOfferId`	string	yes	The bump offer ID to remove
`offerId`	string	yes	The parent offer ID

`omumu_offer_update`

Updates an offer's title, tagline, buttonText, price, or validity dates. Only provided fields are updated; others are preserved.

- **Write category:** standard write

- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`buttonText`	string	no	New button text
`currency`	string	no	New currency code
`offerId`	string	yes	The offer ID
`tagline`	string	no	New tagline
`title`	string	no	New title
`totalAmountCents`	number	no	New total price in cents
`validFrom`	string	no	New start date/time in ISO-8601 format
`validTo`	string	no	New end date/time in ISO-8601 format
`vatAmountCents`	number	no	New VAT amount in cents

CHAPTER 2

The Participant Folder

A mentored programme works because the participant feels known. The mentor remembers what they set out to do, notices when the same thing happens again, and asks about the promise they made last week. AI feedback on its own is the opposite: every hand-in is rated as if it were the first.

The participant folder fixes that. Each participant gets a folder of short, sourced facts about them: their goals, the patterns in their work, their commitments, their context, and your own notes. The AI reads the folder before it rates a hand-in and updates it afterwards. You and your staff curate it on the page you already use for that student, and the participant can read all of it. By week nine the feedback can say "you held the silence you couldn't hold in week one", because the folder remembers week one.

Concepts

A **participant folder** is everything the programme knows about one student on one site: their live **folder facts**, plus their raw history of hand-ins, AI ratings, transcripts and material. A folder belongs to one site and one participant. It is never shared with another site, and it is erased when the participant's account is.

A **folder fact** is one durable statement you can trace back to its source. It has a type, a short name, a one-line description and a body. It also records its

provenance: the hand-in, material or staff member it came from. There are five types:

- **Goal**: what the participant is trying to achieve.
- **Pattern**: something that keeps happening in their work.
- **Commitment**: something they said they would do.
- **Context**: their situation, such as their job, tools and constraints.
- **Mentor note**: something you or your staff want the programme to keep in mind. Only staff write mentor notes; the AI never does.

Facts are never deleted. **Editing** a fact writes a new version that replaces the old one. **Retiring** a fact takes it out of the live folder but keeps it. Both old versions and retired facts stay in a collapsed "Retired facts" section, which shows what replaced each one, who replaced or retired it, and when.

Folder material is anything you add to a folder on the participant's behalf: notes from a mentor call, a question they asked in a webinar, a work conversation, their intake answers. You give it a label and either paste text or upload an audio or video recording. A recording is first transcribed from your site's transcription minutes. The AI then reads the material once and proposes facts from it; material is never scored. The difference from a hand-in is who adds it: the participant hands in their own work, and staff add material.

The **rater** is the AI that rates hand-ins for an assignment with a rating prompt. With the folder, it works like this:

1. **Before rating**, it receives the participant's live facts and their three most recent items of history, next to your rating prompt and the hand-in itself. Facts are always included. If the history is too long, the oldest items are dropped until it fits.
2. **After rating**, its answer has two parts: the feedback the participant sees, as before, and a list of changes to the folder (add, update or retire a fact). Each change points back to the hand-in it came from.

Changes apply at once, with no approval queue. You curate afterwards rather than approving in advance. If a change can't be understood, it is skipped and logged, and the participant still gets their feedback. Rating a hand-in is still one AI request.

Nothing in a folder is hidden from its participant. They can read every live fact and where it came from on their own "Your folder" page, but they can't change anything there. Disputes come to you. If you don't want a participant to read something, don't put it in their folder.

Hands-on

The walkthroughs follow Mira. She has turned her flagship course into a mentored cohort, "Automate Your Week — Coaching Cohort", where participants hand in a short reflection every week. One of them is Ingrid, an office manager at a small accounting firm. Ingrid wants her Friday client-hours report to take thirty minutes instead of three hours.

Mira normally runs her site by talking to her AI assistant, but the folder has **no MCP tools yet**. Curating a folder, adding material and reading a folder all happen in the admin UI, and the walkthroughs below use it. The AI's part runs on its own: the rater and the material reader work in the background whichever way the hand-in or material arrived.

Give a programme a memory

The folder starts working for any assignment that has a **rating prompt**, because that is the assignment the rater runs on. Mira's cohort has one assignment, "Weekly reflection". She sets it up under **Courses → the course → Assignments → New assignment** with this rating prompt:

> You are the coach in "Automate Your Week", a mentored programme where non-programmers learn to automate their weekly work with spreadsheets and AI tools. The participant hands in a weekly reflection. Give warm, specific, honest feedback in at most 180 words. Name one thing that went well and one concrete thing to try next week. Build on what you know about the participant from earlier weeks, and hold them to what they said they would do.

Your prompt doesn't need to explain the folder. The platform adds the folder to the request and asks for the list of changes itself. Write the prompt about the coaching you want, and ask it to build on what it knows. Then publish the assignment.

Start a folder from the intake

Before the first hand-in, Mira gives each participant a starting point. She opens **Students → Ingrid Berg**, scrolls to **Folder** and uses **Add a fact** for what Ingrid's intake form told her:

- a **Goal**, "Weekly report in under 30 minutes", with the details of the Friday routine in the body;
- a **Context** fact, "Office manager at a 12-person accounting firm", recording that she uses Google Sheets daily and has no programming background.

Facts Mira types show "Added by Mira Lund" and read "From: your mentor" on Ingrid's page. Adding, editing and retiring facts needs the **student-management** privilege, the same one that reviewing submissions needs.

Add what you learned on a call

After the kick-off call, Mira doesn't turn her notes into facts by hand. Under **Material → Add material** she picks the label **Mentor call**, pastes her notes and clicks **Add material**. The material shows "Waiting to be read" for a few seconds, then "Read". From her notes the AI added three facts, each marked "Added by the rater" and showing "From: Mentor call" as its source:

- **Context**: the hours come from a Harvest CSV export that Ingrid retypes because the columns are in the wrong order;
- **Pattern**: "Tends to start too big". She wanted to automate invoicing, the report and onboarding all in week one;
- **Commitment**: "Week 1: Build Harvest-importing Google Sheet", working on a copy of the partners' template.

Figure 1: A fact the AI read from Mira's mentor-call notes, with "From: Mentor call" as its source

For a recording, upload it under **Upload a recording** instead of pasting text. It is transcribed first ("Being transcribed", then "Transcribed") and read after

that. The minutes come from your site's transcription pool. If the recording can't be transcribed, the material says why: the month's minutes are used up, the recording is too long, or transcription isn't part of your plan. Once the reason is fixed, **Retry transcription** runs it again.

Watch a hand-in use the folder

Ingrid hands in her week 1 reflection. The sheet works: Friday now takes 70 minutes instead of three hours. She also admits she lost Wednesday afternoon building an invoice tab "because the data was right there". The rater reads her folder before it answers, and the feedback shows it:

> Ingrid, this is real progress: INDEX/MATCH auto-reordering the columns turned three hours into 70 minutes, and you did it on a safe copy just as planned … On the invoicing detour: you caught yourself and named it honestly … a reminder of why we scoped down to one project. Before adding a client summary next week, I'd like you to first tackle the client-name mismatch … That's the last mile of this goal.

It then updated the folder. It recorded the week 1 commitment as done, updated the "Tends to start too big" pattern with the invoicing detour, and added a new commitment for week 2. In the admin panel each of those facts has a **Source** link back to the hand-in.

Keep the folder right

The AI's changes apply without your approval, so curating afterwards is your job. Mira does three things on Ingrid's folder:

- **Edit.** A fact can be nearly right, or worded in a way Mira wouldn't put it to Ingrid, who reads her whole folder. Mira opens **Edit**, rewrites the body and saves. The edit creates a new version, and the old one moves to **Retired facts** marked "Replaced by Tends to start too big · edited by Mira Lund" with the date and time. When the rater updates a fact after a hand-in or a material read, the old version reads "updated by the rater" instead, so her own edits are easy to tell from the AI's.

- **Retire.** The week 1 commitment is done, so she clicks **Retire**. It leaves the live folder at once and stops shaping feedback. It stays under **Retired facts** as "Retired by Mira Lund" with the date and time.
- **Add a mentor note.** She adds a **Mentor note**, "Motivated by getting her Friday back", so the next rating frames Ingrid's next step in minutes saved.

Figure 2: The rater's version of "Tends to start too big" under Retired facts, replaced by Mira's edit

What your participant sees

Ingrid finds **Your folder** at the top of her course list. The page shows her live facts grouped as Goals, Patterns, Commitments, Context and Mentor notes. Each fact shows when it was added and where it came from ("From: your mentor", "From: Mentor call", or a link to the hand-in). Her history is listed below: her hand-ins, linked, and the material Mira added, with its text behind **Show text**. The page has no buttons and no forms. If Ingrid disagrees with something, she takes it up with Mira.

Things to know

- **Every fact is visible to the participant**, including those the AI writes. The AI is told this and writes plain statements of what was said, done or agreed, but it is still worth reading its facts and editing any whose wording you wouldn't use yourself.
- **The rater never writes mentor notes.** Only staff can.
- **The folder is not shared across sites.** A participant who takes a programme on another Omumu site starts with an empty folder there.

- **The cost is small and visible.** Each rating or material read is one AI request. Its input and output tokens are recorded in the outcome log with the result.

Reference

No MCP tools cover this chapter's capabilities yet.

CHAPTER 3

The Agent Surface

The rest of this manual keeps saying "tell your assistant". This chapter is about what makes that sentence true: the single connection point your AI talks to, the credentials that let it in, and the controls that decide — precisely — what it may read, what it may change, and how you take that access back. You don't need to be technical to use any of it; connecting Claude to your site is a copy-paste and two clicks.

If you *are* technical, this chapter is also the receipts. Omumu's agent surface is an MCP server, and every claim below maps to a scope, a permission, or a tool you can check against the generated catalog at the end of this chapter. That catalog — every tool, its parameters, its write category, its access restrictions, and the full permission list — is regenerated from the platform's own capability inventory on every build of this manual, and the build fails if that inventory ever drifts from the code — the catalog below cannot quietly go out of date.

Concepts

The agent surface is one door. Every AI client — Claude on the web, Claude Code in a terminal, a script you wrote — talks to your site through the same endpoint: your site's address plus `/mcp`, speaking the Model Context Protocol (MCP), the open standard AI tools use to operate external systems. There is no second, softer API for agents. An agent acts as an authenticated identity on

your site and passes the same permission checks the admin UI does; the surface is also advertised in machine-readable form — an API catalog at `/.well-known/api-catalog`, an MCP server card at `/.well-known/mcp/server-card.json`, and a plain-text onboarding guide for AI tools at `/llms-onboarding.md` on your site — so an agent pointed at your domain can find the door on its own. MCP access is a plan feature: it must be enabled on your site's subscription tier for the surface to be live.

A **tool** is the unit of capability: one named operation, like `omumu_course_create` or `omumu_offer_list`, with typed parameters. What your assistant "can do" is exactly the set of tools its credential can call — nothing more. Discovery is **fail-closed**: when an agent asks the server what tools exist, it is shown only the ones its credential can actually use. An agent connected with read-only access doesn't see a greyed-out "delete course" button; it doesn't see the tool at all. The full catalog lives in the Reference below, grouped by the chapter of this manual that explains it.

There are **two ways in**, and choosing between them is simple:

An **OAuth connection** is for interactive assistants like Claude. The AI vendor's app registers itself with your site, and you approve the connection on a consent screen *on your own site* — it names the client, lists exactly what it is asking for, and offers Deny and Authorize. Under the hood this is OAuth 2.0 with the current hardening expected of it: authorization code flow with PKCE, and rotating refresh tokens — every refresh issues a new token and retires the old one, and if a retired token is ever replayed (the signature of a stolen token), the site revokes that connection's entire token family on the spot.

An **API key** is for headless use: scripts, scheduled jobs, Claude Code on your laptop. A key looks like `omumu_` followed by a random secret and your site id, is stored only as a salted hash (like a password — the platform itself cannot recover it), and is shown to you exactly once, at creation. Each key carries its own permission picklist, so a reporting script can hold a key that reads courses and nothing else.

Scopes are the language of consent. An OAuth client asks in scopes — `course:read`, `offer:write`, `email:read`, and so on — and each scope maps to a named site permission that every tool checks. Three rules keep this honest. A *write scope implies its read*: `course:write` also grants `course:read`, because editing blind is useless. A client that asks for *nothing* gets the minimum: read access to courses, only. And the scope guarding your buyers' personal data

— `leads:read`, which unlocks names and emails from your opt-in forms — is **never granted by default**: a client must ask for it by name, and you must see it on the consent screen before it's live. That last rule is a recorded architectural decision, not a habit.

Some permissions no agent can hold at all, whichever way it came in: managing user accounts, changing site settings, and managing the API keys themselves are excluded from both the OAuth scope list and the API-key picklist. The last one matters most: an agent cannot mint, widen, or delete credentials — so a misbehaving agent can't promote itself, and revoking it is always a decision only you can make.

Writes are categorized and rate-limited. Every tool carries a write category: read-only tools (listing, getting, reporting) have none; the rest are standard writes or skill writes (skills are the platform's installable agent behaviors — their own chapter). Writes made with an API key pass a per-key, per-category rate limit, so a script stuck in a loop gets slowed down before it can flood your site with half-made courses.

Finally, the catalog also holds a few tools Omumu uses to operate the platform itself. They are not part of this manual, and they are not part of your surface: fail-closed discovery means the agents on your site never even see them.

Stated plainly, the gaps: there are no tools today for site settings, funnels, or analytics dashboards — those are admin-UI work — and no general file upload; the one media tool is `omumu_generate_image`. The catalog below is the honest inventory: if a tool isn't in it, your assistant can't do it, and this manual won't pretend otherwise.

Hands-on

The walkthroughs continue with Mira from the Selling chapter — she runs her course business by talking to her AI assistant. This is the chapter where that setup actually happens.

Connect Claude to your site

Mira uses Claude on the web. In Claude: *Settings* → *Connectors* → *Add custom connector*, paste her site's MCP address — her site URL plus `/mcp` — and confirm.

Claude discovers the OAuth setup on its own and sends her to a consent page *on her own site*, which names the client and lists the scopes it wants. The list is already narrowed to what her account can actually grant, and the defaults are the safe ones from Concepts: no buyer PII, nothing a scope didn't name. She clicks Authorize, and Omumu's tools appear in her Claude conversation.

Two practical notes. Custom connectors require a paid Claude plan (Pro, Max, Team, or Enterprise). And the consent screen is the grant moment: Deny costs nothing and can be retried; Authorize is what creates the connection.

Give a script its own key

For Claude Code — and anything else headless — Mira creates an API key instead. In the admin UI: *Admin → MCP API Keys → create*, where she names the key and ticks the permissions this key should carry — for her weekly sales-summary script, read permissions only. The key is displayed once, on the creation page, with a copy-it-now warning; the same page shows the exact one-line command that registers it with Claude Code:

```
claude mcp add-json omumu '{"type":"http","url":"https://<your
site>/mcp","headers":{"Authorization":"Bearer <your key>"}}'
```

From then on, Claude Code on her laptop operates the site with exactly that key's permissions. Lose the key and there is nothing to recover — the platform kept only the hash — so she creates a new one and deletes the old. One key per purpose is the honest pattern: the key ticked for writes is not the key her reporting script holds.

Stated plainly: creating and managing API keys is **UI-only** by design. There is no tool for it — that's the "no agent manages credentials" rule from Concepts doing its job.

Take an agent's access back

Revoking is the control that makes granting safe, so here is exactly where it stands.

For API keys it is immediate and self-serve: *Admin → MCP API Keys*, delete the key, confirm. The hash is gone and every request bearing that key fails from

that moment. Nothing the agent did survives as access — only as ordinary site data you can still edit.

For OAuth connections, stated plainly: there is **no self-serve revoke button on the Omumu side today**. Removing the connector inside Claude stops Claude from using the connection, and the token machinery limits the damage of a stolen token by itself — refresh tokens rotate on every use, and a replayed old token kills the whole family. But an Omumu-side "disconnect this client" control is admin-UI work that doesn't exist yet; if you need a connection dead from the server side, that is currently a support request. This manual will say otherwise the day the button ships.

A first real conversation

Connected, Mira starts the way anyone should:

> "What can you actually do on my site?"

The assistant answers from its live tool list — which, fail-closed, is exactly what her grant allows. Then she gives it the real test, the same pre-sale move from the Selling chapter, end to end:

> "Create a course called AI Email Triage — one module, five lesson outlines — and put up a 790 NOK early-bird offer for it."

The assistant creates the course (`omumu_course_create`), a module inside it (`omumu_module_create`), five draft lessons (`omumu_lesson_create`), then the offer around it (`omumu_offer_create`, `omumu_offer_add_product`) and reports back with the checkout URL. Mira opens *Admin → Courses* and *Admin → Offers* and finds everything the conversation claimed — the same objects, editable in the same UI, indistinguishable from work done by hand. That is the whole design: the conversation is not a veneer over a separate system. It is your site, through one door, with the keys you chose to hand over.

Reference

Courses & Content

`omumu_answer_create`

Creates an answer option for a question. To enable scoring, link this answer to one or more buckets using omumu_option_point_create after creating the answer.

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`label`	string	yes	The answer label shown to users
`questionId`	string	yes	The question ID
`value`	string	no	The answer value (defaults to label)

`omumu_answer_update`

Updates an answer option's label or value

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`answerId`	string	yes	The answer ID
`label`	string	no	New answer label
`value`	string	no	New answer value

`omumu_assignment_create`

Creates an assignment attached to exactly one scope: a course, a module, or a lesson. Provide exactly one of courseId, moduleId, or lessonId — omitting all or providing more than one is rejected. COURSE scope: the assignment appears on the course homepage and is visible across all its modules and lessons. MODULE scope: the assignment is shown within a specific module and reflects that module's work. LESSON scope: the assignment is tied to a single lesson and reflects its content. New assignments default to DRAFT status; pass status=PUBLISHED to make the assignment immediately visible to students. Requires the MCP_COURSES feature (Starter tier or above).

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`asksForLight`	boolean	no	Check-in: every hand-in must include the student's own Light — green, yellow or red. Defaults to false.
`courseId`	string	no	Attach the assignment to this course (COURSE scope). Must not be combined with moduleId or lessonId.
`description`	string	no	The assignment instructions or description (Markdown)
`lessonId`	string	no	Attach the assignment to this lesson (LESSON scope). The owning course is derived automatically. Must not be combined with courseId or moduleId.
`moduleId`	string	no	Attach the assignment to this module (MODULE scope). The owning course is derived automatically. Must not be combined with courseId or lessonId.
`ratingPrompt`	string	no	Prompt shown to instructors when rating a submission (optional)
`status`	string	no	Initial status: DRAFT (default, not visible to students) or PUBLISHED (visible)
`title`	string	yes	The assignment title

`omumu_assignment_list`

Returns all assignments (DRAFT and PUBLISHED) for a course, with each assignment's scope kind (COURSE, MODULE, or LESSON) and the scope's id stated explicitly. Requires the MCP_COURSES feature (Starter tier or above).

- **Write category:** read-only

- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course whose assignments to list

`omumu_assignment_update`

Updates an existing assignment's title, description, ratingPrompt, status, and/or scope. Provide exactly one of courseId, moduleId, or lessonId to set the new scope. The new scope must belong to the same course as the assignment. Fields not provided are left unchanged. Requires the MCP_COURSES feature (Starter tier or above).

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`asksForLight`	boolean	no	Check-in: every hand-in must include the student's own Light — green, yellow or red. Omit to leave unchanged.
`assignmentId`	string	yes	The assignment ID to update
`courseId`	string	no	New COURSE scope. Must not be combined with moduleId or lessonId.
`description`	string	no	New assignment instructions (Markdown)
`lessonId`	string	no	New LESSON scope. Must not be combined with courseId or moduleId.
`moduleId`	string	no	New MODULE scope. Must not be combined with courseId or lessonId.
`ratingPrompt`	string	no	New rating prompt for instructors
`status`	string	no	New status: DRAFT (hidden from students) or PUBLISHED (visible)
`title`	string	no	New assignment title

`omumu_bucket_create`

Creates a result bucket (outcome category) for the quiz. Users are sorted into the bucket with the highest score based on their answers. After creating buckets, link answers to them using omumu_option_point_create. Each bucket can trigger actions: send email sequence, grant course access, redirect to URL.

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	Bucket description
`name`	string	yes	The bucket name (e.g., 'Beginner', 'Advanced')
`quizId`	string	yes	The quiz ID

`omumu_bucket_update`

Updates a bucket's properties including actions (email sequence, course, redirect)

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`bucketId`	string	yes	The bucket ID
`courseId`	string	no	Course to grant access to
`description`	string	no	New description
`emailSequenceId`	string	no	Email sequence to subscribe user to
`name`	string	no	New bucket name
`redirectUrl`	string	no	URL to redirect to after quiz
`successMessage`	string	no	Message shown after quiz completion
`tagName`	string	no	Tag to apply to user

`omumu_checklist_batch_create`

Creates multiple checklist items for a lesson in one call (max 100). Atomic — all items are created or none. Useful for AI workflows that generate complete checklists at once.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`items`	array	yes	Array of objects with 'description' (required, max 1000 chars) and 'xp' (optional, 0-10000, default 0)
`lessonId`	string	yes	The lesson ID

`omumu_checklist_item_create`

Adds a new checklist item to a lesson. Checklist items are actionable tasks that students complete to earn XP. Items are auto-appended at the end of the list.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`description`	string	yes	The checklist item text, max 1000 chars (what the student should do)
`lessonId`	string	yes	The lesson ID to add the checklist item to
`xp`	integer	no	XP points awarded when completed, 0-10000 (default 0)

`omumu_checklist_item_delete`

Permanently deletes a checklist item and any associated user progress. Remaining items are re-sorted automatically.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`checklistItemId`	string	yes	The checklist item ID to delete

`omumu_checklist_item_list`

Lists all checklist items for a specific lesson, ordered by sort order.

- **Write category:** read-only

- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`lessonId`	string	yes	The lesson ID

`omumu_checklist_item_reorder`

Changes the sort order of a checklist item within its lesson.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`checklistItemId`	string	yes	The checklist item ID
`newSortOrder`	integer	yes	The new sort order position (0-based)

`omumu_checklist_item_update`

Updates the description and/or XP value of an existing checklist item.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`checklistItemId`	string	yes	The checklist item ID
`description`	string	no	New description text, max 1000 chars (optional)
`xp`	integer	no	New XP value, 0-10000 (optional)

`omumu_course_create`

Creates a new course in the user's site with the specified title and tagline. After creating, add modules (omumu_module_create) and lessons (omumu_lesson_create) to build the course structure. To grant access via an opt-in form, pass the course ID to omumu_optinform_update(courseId).

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`tagline`	string	no	The course tagline
`title`	string	yes	The course title

`omumu_course_delete`

Permanently deletes a course and all its content

- **Write category:** standard write
- **Access:** requires the `COURSE_DELETE` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course ID to delete

`omumu_course_dna_build`

Materializes the reviewed structure into the offer's course: modules and lessons through the normal course machinery, each lesson's body seeded from its brief (promise, extracted expertise, cited facts, beats) — never AI-drafted prose. Blocks with the unmapped promise names if the structure fails the promise-coverage gate. Links the Course DNA to the course and keeps the buyer's entitlement pointing at it.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`id`	string	yes	The Course DNA id

`omumu_course_dna_create`

Starts the offer-driven course document (ADR-0037) against a LAUNCHED Offer DNA. Strictly offer-first: the contract — touchstone, dream outcome, stack, bonuses, objections, verbatim pains, format intent, frozen price — is read from the linked Offer DNA + Offer, never re-invented. One course per offer: if a Course DNA already exists for that Offer DNA, it is returned (resumed) instead of duplicated.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`offerDnaId`	string	yes	The launched Offer DNA to deliver on (see omumu_offer_dna_list)

`omumu_course_dna_extraction_save`

Persists the extraction — the creator's OWN expertise, harvested by interviewing them: their method, its steps in order, the mistakes learners make, their stories, examples, and existing materials. Interview the creator and record their answers; NEVER invent domain expertise, facts, steps, or stories the creator did not give you — the creator's knowledge is the content source. Re-saving replaces the section; it stays reviewable until the structure step consumes it.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`examples`	array	no	Concrete examples the creator gave
`existingMaterials`	string	no	Materials the creator already has (notes, decks, recordings)
`id`	string	yes	The Course DNA id
`learnerMistakes`	array	no	Mistakes the creator has seen learners make
`method`	string	no	The creator's method or approach, in their own words
`steps`	array	no	The method's steps, in order, as the creator described them
`stories`	array	no	The creator's stories — wins, failures, before/after moments

`omumu_course_dna_get`

Returns the full Course DNA document, contract first.

- **Write category:** read-only

- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`id`	string	yes	The Course DNA id

`omumu_course_dna_list`

Lists the site's Course DNA documents, newest first.

- **Write category:** read-only
- **Access:** requires the `COURSE_READ` permission

No parameters.

`omumu_course_dna_research_run`

OPTIONAL and off by default — call it only when the creator explicitly wants cited external facts (statistics, standards, tool specifics) for their lessons; calling this tool is the opt-in. One AI run billed to the daily budget, with web search constrained to a server-controlled high-trust source tier. Every finding carries its source URL. The creator's own extraction stays the content source; research only supplements it. Re-running replaces the section.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`id`	string	yes	The Course DNA id

`omumu_course_dna_structure_run`

Proposes the transformation-based outline (one AI run, billed to the daily budget) from the contract and the creator's extraction, sized by the format intent. Every contract stack item must be covered by a lesson or explicitly deferred — a deterministic gate blocks the build otherwise. Requires the extraction to be saved first. Re-running replaces the outline. Review the outline with the creator before building.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`id`	string	yes	The Course DNA id

`omumu_course_duplicate`

Creates a complete copy of an existing course including all modules and lessons

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course ID to duplicate

`omumu_course_get`

Retrieves detailed information about a specific course

- **Write category:** read-only
- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course ID
`includeModules`	boolean	no	Whether to include modules and lessons

`omumu_course_list`

Lists all courses in the user's site with optional module inclusion

- **Write category:** read-only
- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`includeModules`	boolean	no	Whether to include modules in the response

`omumu_course_update`

Updates the basic information of a course

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course ID
`description`	string	no	New course description
`gatingMode`	string	no	Course gating mode: OPEN (default — course access opens every module) or SEQUENTIAL (modules unlock via module-scoped access entries, e.g. the unlock_module automation action)
`tagline`	string	no	New course tagline
`title`	string	no	New course title
`watchThresholdPercent`	integer	no	How much of a video counts as watched, in percent. One of 25, 50, 75, 90, 100 (default 75)

`omumu_generate_image`

Generates an image from a text prompt using AI and stores it as a resource. Optionally attach the generated image directly to a course, module, or lesson. When targeting an entity, aspect ratio defaults to 16:9 (ideal for course illustrations).

- **Write category:** standard write
- **Access:** requires the `MEDIA_GENERATE` permission

Parameter	Type	Required	Description
`aspectRatio`	string	no	Aspect ratio: 1:1, 16:9, 9:16, 4:3, 3:4. Default: 16:9 when targeting, 1:1 otherwise.
`highQuality`	boolean	no	Use high quality model (slower, requires Premium/Enterprise tier). Default: false
`prompt`	string	yes	Text description of the image to generate. Be descriptive for best results.
`targetId`	string	no	ID of the entity to attach the image to. Required if targetType is provided.
`targetType`	string	no	Entity type to attach image to: 'course', 'module', or 'lesson'. If provided, targetId is also required.

`omumu_lesson_create`

Creates a new lesson within a module. Returns a compact summary {id, title, sortOrder} only — the submitted description is not echoed back, to keep responses small when authoring long-form content.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	The lesson description
`moduleId`	string	yes	The module ID
`title`	string	yes	The lesson title

`omumu_lesson_resource_create`

Creates a new content block within a lesson. Content can be Markdown or QuillDelta JSON (auto-detected). Optionally attaches an uploaded SiteResource (e.g. a video) via resourceId. Use this to add rich text content to lessons - the 'meat' of course content.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`content`	string	yes	The content in Markdown or QuillDelta JSON format. Markdown supports: # headers, **bold**, *italic*, links, - bullets, 1. numbered lists, > blockquotes, `code`
`lessonId`	string	yes	The lesson ID to add content to
`resourceId`	string	no	Optional SiteResource ID (e.g. an uploaded video) to attach to this lesson part. Must belong to the same site as the lesson.
`title`	string	yes	Title of the content block

`omumu_lesson_resource_delete`

Permanently deletes a content block from a lesson.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`lessonResourceId`	string	yes	The lesson resource ID to delete

`omumu_lesson_resource_list`

Lists all content blocks for a specific lesson.

- **Write category:** read-only
- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`lessonId`	string	yes	The lesson ID

`omumu_lesson_resource_update`

Updates an existing content block. Content can be Markdown or QuillDelta JSON (auto-detected). Optionally replaces the attached SiteResource. Pass an empty string for resourceId to clear the attachment.

- **Write category:** standard write

- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`content`	string	no	New content in Markdown or QuillDelta JSON (optional)
`lessonResourceId`	string	yes	The lesson resource ID
`resourceId`	string	no	New SiteResource ID to attach (optional). Pass empty string to clear the current attachment. Must belong to the same site as the lesson.
`title`	string	no	New title (optional)

`omumu_lesson_update`

Updates a lesson's title and/or description. Description can be Markdown or QuillDelta JSON (auto-detected). Returns a compact summary {id, title, sortOrder} only — the submitted description is not echoed back, to keep responses small when authoring long-form content.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	New lesson description (Markdown or QuillDelta JSON)
`lessonId`	string	yes	The lesson ID
`title`	string	no	New lesson title

`omumu_module_copy`

Copies a module (including all its lessons and resources) from its current course into a different target course. Both the source module's course and the target course must belong to the user's site.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`moduleId`	string	yes	The ID of the module to copy
`targetCourseId`	string	yes	The ID of the course to copy the module into

`omumu_module_create`

Creates a new module within a course

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course ID
`description`	string	no	The module description
`title`	string	yes	The module title

`omumu_module_list`

Lists all modules in a specific course with optional lesson inclusion

- **Write category:** read-only
- **Access:** requires the `COURSE_READ` permission

Parameter	Type	Required	Description
`courseId`	string	yes	The course ID
`includeLessons`	boolean	no	Whether to include lessons in each module

`omumu_module_update`

Updates a module's title and/or description. Description can be Markdown or QuillDelta JSON (auto-detected).

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	New module description (Markdown or QuillDelta JSON)
`moduleId`	string	yes	The module ID
`title`	string	no	New module title

`omumu_option_point_batch`

Efficiently set up quiz scoring by creating or updating multiple option points in a single call. If an answer-bucket pair already exists, updates its weight. Use this after creating questions, answers, and buckets to configure the scoring matrix. Example: [{answerId: 'a1', bucketId: 'b1', weight: 2.0}, {answerId: 'a1', bucketId: 'b2', weight: 1.0}]

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`optionPoints`	array	yes	Array of option points, each with: answerId (required), bucketId (required), weight (optional, default 1.0)

`omumu_option_point_create`

Creates an option point linking an answer to a bucket with a weight. When a user selects this answer, the weight is added to the bucket's score. After all questions, the bucket with the highest total score becomes the user's result. Example: Answer 'I exercise daily' → Bucket 'Fitness Enthusiast' with weight 2.0.

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`answerId`	string	yes	The answer ID to link
`bucketId`	string	yes	The bucket ID to assign points to
`weight`	number	no	Point weight (default: 1.0). Higher weights have more influence on final bucket selection.

`omumu_option_point_delete`

Removes the link between an answer and a bucket, so selecting this answer no longer contributes to that bucket's score

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`optionPointId`	string	yes	The option point ID to delete

`omumu_option_point_list`

Lists all option points for a specific question, showing which answers are linked to which buckets and their weights

- **Write category:** read-only
- **Access:** requires the `QUIZ_READ` permission

Parameter	Type	Required	Description
`questionId`	string	yes	The question ID

`omumu_option_point_update`

Updates an option point's weight to adjust scoring influence

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`optionPointId`	string	yes	The option point ID
`weight`	number	no	New weight value

`omumu_question_create`

Creates a new question in a quiz. After creating a question, add answer options with omumu_answer_create. RADIO type shows single-choice radio buttons, CHECKBOX allows multiple selections.

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`header`	string	yes	The question text
`quizId`	string	yes	The quiz ID
`type`	string	no	Question type: RADIO, CHECKBOX, TEXTAREA, RADIO_OTHER, CHECKBOX_OTHER

`omumu_question_update`

Updates a question's header or type

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`header`	string	no	New question text
`questionId`	string	yes	The question ID
`type`	string	no	New question type

`omumu_quiz_create`

Creates a new quiz (question set) for lead generation or segmentation. Quizzes can optionally link to opt-in forms for email capture: preQuizOptinFormId shows an email capture form before the quiz starts (for non-logged-in users), postQuizOptinFormId shows one after quiz completion. After creating a quiz, add questions with omumu_question_create, then add answer options with omumu_answer_create, create result buckets with omumu_bucket_create, and finally link answers to buckets using omumu_option_point_create.

- **Write category:** standard write
- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`label`	string	yes	The quiz name/label
`purpose`	string	no	Purpose: LEAD_GENERATION or SEGMENTATION

`omumu_quiz_get`

Gets a quiz with optional questions and result buckets. Returns preQuizOptinFormId and postQuizOptinFormId if linked opt-in forms are configured for email capture. To see the complete quiz scoring configuration, set includeQuestions=true, includeBuckets=true, and includeOptionPoints=true. Option points link answers to buckets with weights - the bucket with the highest total weight from selected answers wins.

- **Write category:** read-only
- **Access:** requires the `QUIZ_READ` permission

Parameter	Type	Required	Description
`includeBuckets`	boolean	no	Include result buckets
`includeOptionPoints`	boolean	no	Include option points (answer-to-bucket scoring weights). Requires questions to be fetched.
`includeQuestions`	boolean	no	Include questions and answers
`quizId`	string	yes	The quiz ID

`omumu_quiz_list`

Lists all quizzes for the current site

- **Write category:** read-only
- **Access:** requires the `QUIZ_READ` permission

No parameters.

`omumu_quiz_update`

Updates a quiz's properties including label and linked opt-in forms for email capture. Pre-quiz opt-in forms appear before the quiz starts for non-logged-in users. Post-quiz opt-in forms appear after completion. Set to null to remove the link.

- **Write category:** standard write

- **Access:** requires the `QUIZ_WRITE` permission

Parameter	Type	Required	Description
`label`	string	no	New label
`postQuizOptinFormId`	string	no	ID of opt-in form to show after quiz completion (for non-logged-in users), or null to remove
`preQuizOptinFormId`	string	no	ID of opt-in form to show before quiz (for non-logged-in users), or null to remove
`quizId`	string	yes	The quiz ID

`omumu_resource_delete`

Permanently deletes a SiteResource. Fails if the resource is currently referenced by any lesson, page, or other entity — remove those references first.

- **Write category:** standard write
- **Access:** requires the `RESOURCE_DELETE` permission

Parameter	Type	Required	Description
`resourceId`	string	yes	The resource ID to delete

`omumu_resource_get`

Returns the full metadata for a single SiteResource, including CDN URI, thumbnail, MIME type, file size, and video processing status (when applicable).

- **Write category:** read-only
- **Access:** requires the `RESOURCE_READ` permission

Parameter	Type	Required	Description
`resourceId`	string	yes	The resource ID

`omumu_resource_list`

Lists uploaded files (videos, images, PDFs, etc.) on the current site. Optionally filter by a MIME type prefix (e.g. 'video/' or 'image/png') and paginate with limit/offset. Returns total count so callers can implement 'load more' paging.

- **Write category:** read-only
- **Access:** requires the `RESOURCE_READ` permission

Parameter	Type	Required	Description
`limit`	number	no	Maximum resources to return (default: 25, max: 100)
`mimeTypePrefix`	string	no	Optional prefix match against the resource's MIME type, e.g. 'video/' to return all videos or 'image/png' to return only PNG images. Omit to list all types.
`offset`	number	no	Number of resources to skip for pagination (default: 0)

`omumu_resource_update`

Updates the title and/or description of a SiteResource. Does not change the underlying file — re-upload for that. At least one of title/description must be provided.

- **Write category:** standard write
- **Access:** requires the `RESOURCE_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	New description (optional)
`resourceId`	string	yes	The resource ID
`title`	string	no	New title (optional)

`omumu_resource_upload_video_presign`

Creates a SiteResource for a video (videoProcessingStatus QUEUED) and returns a presigned TUS (tus.io) upload the caller PATCHes bytes to directly — the video never passes through the MCP server or the JSON-RPC channel. Use the returned resourceId right away with omumu_lesson_resource_create or omumu_set_image; poll omumu_resource_get for videoProcessingStatus once the upload finishes and Bunny's webhook reconciles it. To perform the upload: POST to the returned 'endpoint' with headers AuthorizationSignature, AuthorizationExpire, VideoId (the 'guid' field), LibraryId, Upload-Length (the file size), and Tus-Resumable: 1.0.0 — Bunny responds with a Location header, which is the URL to PATCH the file bytes to (Content-Type: application/offset+octet-stream, Upload-Offset: 0, Tus-Resumable: 1.0.0).

- **Write category:** standard write
- **Access:** requires the `RESOURCE_WRITE` permission

Parameter	Type	Required	Description
`filename`	string	yes	Original file name of the video, e.g. 'lesson-1.mp4'
`filetype`	string	yes	MIME type of the video: video/mp4, video/webm, video/ogg, video/quicktime, or video/x-matroska
`length`	number	yes	Total file size in bytes

`omumu_set_image`

Attach an existing image resource to a course, module, or lesson. Use this to reuse an already generated image or to change an entity's image.

- **Write category:** standard write
- **Access:** requires the `COURSE_WRITE` permission

Parameter	Type	Required	Description
`resourceId`	string	yes	ID of the image resource to attach
`targetId`	string	yes	ID of the entity to set the image on
`targetType`	string	yes	Entity type: 'course', 'module', or 'lesson'

Selling

`omumu_offer_add_bump`

Adds a bump offer to an existing offer. The bump appears as a checkbox on the checkout page. Both offers must belong to the same site. An offer cannot be its own bump.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`bumpOfferId`	string	yes	The bump offer ID to add
`offerId`	string	yes	The parent offer ID

`omumu_offer_add_product`

Adds a product (course or skill) to an existing offer. The offer must not be live or ended. Subscription plans (SAAS) are added via omumu_offer_add_subscription_plan instead.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID
`productId`	string	yes	The product ID to add — a course id or a skill id (the meta_product id, which equals the underlying course/skill id)

`omumu_offer_add_subscription_plan`

Adds a subscription plan (Omumu platform access) to an existing offer. When the offer is purchased, the buyer gets a new Omumu site provisioned on the plan's tier. Use SAAS_BASIC for the Omumu Basic plan.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID
`subscriptionPlanId`	string	yes	The subscription plan ID (e.g., SAAS_BASIC)

`omumu_offer_create`

Creates a purchasable offer with title, price, courses, and payment providers. The offer starts in 'scheduled' status. Use validFrom/validTo to control availability. Returns the offer with its checkout URL.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`buttonText`	string	no	Text for the purchase button (e.g., 'Buy now')
`courseIds`	array	no	Array of course ID strings to include in the offer
`currency`	string	no	Three-letter currency code (default: NOK)
`descriptionPageId`	string	no	Page ID for the offer description shown at checkout
`paymentProviders`	array	no	Array of payment provider names: STRIPE, VIPPS, INVOICE_REQUEST, NONE
`subscriptionPlanId`	string	no	Subscription plan ID to include (e.g., SAAS_BASIC for Omumu Basic). Bundles Omumu platform access into the offer.
`tagline`	string	no	A short tagline or subtitle
`thankYouPageId`	string	no	Page ID for the post-purchase thank-you page
`title`	string	yes	The offer title
`totalAmountCents`	number	yes	Total price in cents (e.g., 4900 for 49.00 NOK)
`validFrom`	string	no	Start date/time in ISO-8601 format (e.g., 2026-04-01T00:00:00)
`validTo`	string	no	End date/time in ISO-8601 format
`vatAmountCents`	number	no	VAT amount in cents (included in totalAmountCents)

`omumu_offer_delete`

Deletes an offer. Only offers that have not been sold can be deleted.

- **Write category:** standard write
- **Access:** requires the `OFFER_DELETE` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID to delete

`omumu_offer_get`

Gets an offer by ID with full details including checkout URL, products, pricing, and payment providers

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

Parameter	Type	Required	Description
`offerId`	string	yes	The offer ID

`omumu_offer_grant_access`

Grants a user access to all products in an offer WITHOUT requiring payment. This bypasses the normal checkout/payment flow entirely. The user receives the same access as if they had paid: STUDENT role, product access entries, welcome email, and offer-tag execution. An offer containing a SAAS plan can only be granted by Omumu. The user must already exist on the same site as the offer.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`email`	string	yes	Email of the user to grant access to. Must already exist on the same site as the offer
`message`	string	no	Optional free-text reason for the grant (e.g. 'Demo account for Ole-Arvid'). Recorded in the audit log only, not stored on the access entry. Max 500 characters
`offerId`	string	yes	The offer to grant access to

`omumu_offer_list`

Lists all offers for the current site with their status (scheduled/live/ended), products, and pricing

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

No parameters.

`omumu_offer_remove_bump`

Removes a bump offer from an existing offer.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`bumpOfferId`	string	yes	The bump offer ID to remove
`offerId`	string	yes	The parent offer ID

`omumu_offer_update`

Updates an offer's title, tagline, buttonText, price, or validity dates. Only provided fields are updated; others are preserved.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`buttonText`	string	no	New button text
`currency`	string	no	New currency code
`offerId`	string	yes	The offer ID
`tagline`	string	no	New tagline
`title`	string	no	New title
`totalAmountCents`	number	no	New total price in cents
`validFrom`	string	no	New start date/time in ISO-8601 format
`validTo`	string	no	New end date/time in ISO-8601 format
`vatAmountCents`	number	no	New VAT amount in cents

Pages & Sales Pages

`omumu_page_create`

Creates a new page. Two modes:

Slot page (recommended for sales pages): Pass ‘template’ (discriminator from omumu_template_list) and ‘slotJson’ (whole-document slot JSON validated against that template). For SALES templates, also pass ‘offerId’. Wiring fields (offerId, purpose) must be tool parameters — never inside slot contents.

Classic page: Pass ‘contents’ with optional ‘encoding’ (QUILL_JSON or MANAGED_HTML). MANAGED_HTML is available but de-emphasised for selling pages — prefer slot pages instead.

- **Write category:** standard write
- **Access:** requires the `PAGE_WRITE` permission

Parameter	Type	Required	Description
`contents`	string	no	Classic page content. Raw HTML when encoding is MANAGED_HTML, Quill JSON or markdown otherwise. Not used when creating slot pages.
`encoding`	string	no	Content encoding for classic pages: QUILL_JSON (default) or MANAGED_HTML. Not used when creating slot pages.
`offerId`	string	no	Offer ID this sales page sells. Required for SALES templates; must be a valid offer belonging to this site. Never pass inside slot contents.
`slotJson`	string	no	Whole-document slot JSON validated against the named template. Required when 'template' is provided.
`slug`	string	yes	The page URL slug (e.g., 'my-course-launch')
`template`	string	no	Template discriminator from omumu_template_list (e.g. 'sales-cold-traffic'). Required for slot page creation; causes slotJson validation against the named template.
`title`	string	no	The page title
`type`	string	no	Page type: blog, frontpage, userpage, terms, privacy, cookies, template, block, offer, thank_you

`omumu_page_delete`

Deletes a page by ID

- **Write category:** standard write
- **Access:** requires the `PAGE_WRITE` permission

Parameter	Type	Required	Description
`pageId`	string	yes	The page ID to delete

`omumu_page_get`

Gets a page by ID or slug. At least one of pageId or slug must be provided.

- **Write category:** read-only
- **Access:** requires the `PAGE_READ` permission

Parameter	Type	Required	Description
`pageId`	string	no	The page ID (provide this or slug)
`slug`	string	no	The page slug (provide this or pageId)

`omumu_page_list`

Lists all pages for the current site, optionally filtered by type

- **Write category:** read-only
- **Access:** requires the `PAGE_READ` permission

Parameter	Type	Required	Description
`excludeOfferPages`	boolean	no	Exclude auto-generated offer pages
`type`	string	no	Filter by page type: blog, frontpage, userpage, terms, privacy, cookies, template, block, offer, thank_you

`omumu_page_update`

Updates a page. For slot pages (encoding SLOT_JSON), pass ‘slotJson’ with the full updated slot document — wiring (offerId, purpose) is immutable on update and is not re-validated here. For classic pages, pass any combination of ‘title’, ‘tagline’, ‘metaDescription’, ‘contents’, ‘image’, or ‘slug’.

- **Write category:** standard write
- **Access:** requires the `PAGE_WRITE` permission

Parameter	Type	Required	Description
`contents`	string	no	New content for classic pages. Format depends on page encoding: raw HTML for MANAGED_HTML pages, Quill JSON for QUILL_JSON pages.
`image`	string	no	New header image URL
`metaDescription`	string	no	New meta description for SEO
`pageId`	string	yes	The page ID
`rootPublished`	boolean	no	Serve the page at the site root (/<slug>) in addition to /page/<slug>. Reserved platform names (admin, terms, courses, …) are refused. Both addresses keep working; rendered links, canonical metadata, and the sitemap use the root URL. Omit to leave unchanged.
`slotJson`	string	no	Full updated slot JSON document for slot pages (encoding SLOT_JSON). The template discriminator is read from the existing slot document. Wiring is immutable.
`slug`	string	no	New URL slug (changes the page URL)
`tagline`	string	no	New tagline
`title`	string	no	New title

`omumu_template_list`

Returns all registered page templates with their section layouts and per-section JSON schema. Call this tool first when building a slot page so you know which template discriminator to pass to omumu_page_create and what slot fields each section expects. Each entry includes: ‘name’ (the discriminator), ‘purpose’ (e.g. SALES), ‘sections’, and ‘schema’.

- **Write category:** read-only
- **Access:** requires the `PAGE_READ` permission

No parameters.

Funnels

`omumu_funnel_add_upsell`

Scaffolds a one-click upsell on an existing funnel. Inserts the upsell step between the checkout step and its current successor, and optionally wires a downsell on decline. Requires the UPSELLS feature on the site's tier. Supply either an existing published offer id or name+price to create one inline.

- **Write category:** standard write
- **Access:** requires the `FUNNEL_WRITE` permission

Parameter	Type	Required	Description
`downsellCurrency`	string	no	Currency code for the new downsell offer (default NOK). Used with downsellName.
`downsellEnabled`	boolean	no	Set to true to add a downsell step on decline.
`downsellName`	string	no	Name for a new downsell offer to create inline. Used when downsellEnabled is true.
`downsellOfferId`	string	no	ID of an existing published offer to use as the downsell. Used when downsellEnabled is true.
`downsellPriceCents`	number	no	Price in cents for the new downsell offer. Used with downsellName.
`funnelId`	string	yes	The funnel to add the upsell to
`seedLanguage`	string	no	Language for the seeded page copy: 'en' (default) or 'nb'.
`upsellCurrency`	string	no	Currency code for the new upsell offer (default NOK). Used with upsellName.
`upsellName`	string	no	Name for a new upsell offer to create inline. Required when upsellOfferId is absent.
`upsellOfferId`	string	no	ID of an existing published offer to use as the upsell. Mutually exclusive with upsellName/upsellPriceCents.
`upsellPriceCents`	number	no	Price in cents for the new upsell offer (e.g. 4900 = 49.00 NOK). Required with upsellName.

`omumu_funnel_get`

Gets a funnel by id with the full branched step graph: each step's derived kind (SALES, CHECKOUT, UPSELL, THANK_YOU, OPT_IN, PAGE_VIEW), accept edge (nextStepId), decline edge (declineStepId, UPSELL steps only), and the entry step id.

- **Write category:** read-only
- **Access:** requires the `FUNNEL_READ` permission

Parameter	Type	Required	Description
`funnelId`	string	yes	The funnel id

`omumu_funnel_list`

Lists all funnels for the current site with their id, name, and offer anchor id.

- **Write category:** read-only
- **Access:** requires the `FUNNEL_READ` permission

No parameters.

Audience & Email

`omumu_email_followup_update`

Updates a follow-up message in an email sequence. All fields except followUpId are optional.

- **Write category:** standard write
- **Access:** requires the `EMAIL_SEQUENCE_WRITE` permission

Parameter	Type	Required	Description
`content`	string	no	New content in Markdown or QuillDelta JSON
`delay`	string	no	New delay from the first email (absolute, NOT relative to previous). Duration string. Accepts ISO-8601 (e.g. PT4H, P1D, P2DT12H, PT0S) or shorthand <integer><m|h|d|w> (e.g. 30m, 4h, 7d, 2w).
`followUpId`	string	yes	The follow-up ID to update
`subject`	string	no	New subject line

`omumu_email_inbox_get`

Retrieves full details of a specific email including headers, body content (text and/or HTML), attachments, and metadata. Use omumu_email_inbox_list first to find email IDs.

- **Write category:** read-only
- **Access:** requires the `EMAIL_INBOX_READ` permission

Parameter	Type	Required	Description
`emailId`	string	yes	The email ID to retrieve

`omumu_email_inbox_list`

Lists received emails in the inbox with optional filtering by status or search query. Returns email summaries (id, sender, subject, status, date) with pagination. Also returns unread count for quick monitoring.

- **Write category:** read-only
- **Access:** requires the `EMAIL_INBOX_READ` permission

Parameter	Type	Required	Description
`limit`	number	no	Maximum emails to return per page (default: 25, max: 100)
`page`	number	no	Page number for pagination (default: 1)
`search`	string	no	Search in sender address, sender name, subject, or recipient address
`status`	string	no	Filter by status: UNREAD, READ, ARCHIVED, HELD, or ALL (default: the inbox, which excludes deleted and held email)

`omumu_email_sequence_create`

Creates a new email sequence with an initial email and optional follow-up messages. Content can be Markdown or QuillDelta JSON (auto-detected). After creating, link it to an opt-in form via omumu_optinform_update(emailSequenceId) to trigger the sequence when users submit the form. To send a single one-off email to a contact from an automation, create a sequence containing just one message with initialDelay PT0S and start it with the automation's start_sequence action.

- **Write category:** standard write
- **Access:** requires the `EMAIL_SEQUENCE_WRITE` permission

Parameter	Type	Required	Description
`content`	string	yes	Email content in Markdown or QuillDelta JSON
`followUps`	array	no	Array of follow-up messages with delay (offset from the first email, NOT relative to previous), subject, and content. Duration string. Accepts ISO-8601 (e.g. PT4H, P1D, P2DT12H, PT0S) or shorthand <integer><m|h|d|w> (e.g. 30m, 4h, 7d, 2w).
`initialDelay`	string	no	Delay before sending the first email. Duration string. Accepts ISO-8601 (e.g. PT4H, P1D, P2DT12H, PT0S) or shorthand <integer><m|h|d|w> (e.g. 30m, 4h, 7d, 2w). Default: PT0S.
`name`	string	yes	Sequence name (unique identifier)
`subject`	string	yes	Subject line for the initial email

`omumu_email_sequence_get`

Retrieves a specific email sequence with all follow-up messages.

- **Write category:** read-only
- **Access:** requires the `EMAIL_SEQUENCE_READ` permission

Parameter	Type	Required	Description
`sequenceId`	string	yes	The sequence ID to retrieve

`omumu_email_sequence_list`

Lists all email sequences for the current site with their follow-up messages.

- **Write category:** read-only
- **Access:** requires the `EMAIL_SEQUENCE_READ` permission

No parameters.

`omumu_email_sequence_update`

Updates an existing email sequence's properties. All fields except sequenceId are optional.

- **Write category:** standard write
- **Access:** requires the `EMAIL_SEQUENCE_WRITE` permission

Parameter	Type	Required	Description
`content`	string	no	New content in Markdown or QuillDelta JSON
`initialDelay`	string	no	New delay from the subscription trigger. Duration string. Accepts ISO-8601 (e.g. PT4H, P1D, P2DT12H, PT0S) or shorthand <integer><m|h|d|w> (e.g. 30m, 4h, 7d, 2w).
`name`	string	no	New sequence name
`sequenceId`	string	yes	The sequence ID to update
`subject`	string	no	New subject line for initial email

`omumu_invite_get`

Gets a single Invite by ID with full config, live window state (accepted of quota, next boundary), and waitlist size.

- **Write category:** read-only
- **Access:** requires the `INVITE_READ` permission

Parameter	Type	Required	Description
`inviteId`	string	yes	The Invite ID

`omumu_invite_launch`

Creates a complete Honest Scarcity Invite in one call: a course (or attaches an existing one), three seeded email sequences (delivery, waitlist-welcome, reopen), an opt-in form wired to the course grant and delivery sequence, the Invite itself, the waitlist tag action on the reopen sequence, and a live invite page — all wired and ready to publish. Returns the public page URL and the slot editor path so you can customize copy immediately. Cadence values: HOUR, DAY, WEEK, MONTH. openingDayOfWeek required for WEEK cadence

(e.g. MONDAY). openingDayOfMonth required for MONTH cadence (1-31). openingTime format: HH:MM (24h); defaults to 09:00. zone: IANA timezone id (e.g. 'Europe/Oslo'); defaults to Europe/Oslo. seedLanguage: 'nb' for Norwegian Bokmål, 'en' (or omit) for English.

- **Write category:** standard write
- **Access:** requires the `INVITE_WRITE` permission

Parameter	Type	Required	Description
`cadence`	string	yes	Reset cadence: HOUR, DAY, WEEK, or MONTH
`courseId`	string	no	Existing course ID to attach; omit to create a new empty course
`name`	string	yes	Human-readable name for the Invite (also used as the page headline)
`openingDayOfMonth`	number	no	Day of month for MONTH cadence (1-31)
`openingDayOfWeek`	string	no	Day of week for WEEK cadence (e.g. MONDAY)
`openingTime`	string	no	Opening time in HH:MM (24h); defaults to 09:00
`quota`	number	yes	Slots available per window
`seedLanguage`	string	no	'nb' for Norwegian Bokmål, 'en' or omit for English
`zone`	string	no	IANA timezone id (e.g. 'Europe/Oslo'); defaults to Europe/Oslo

`omumu_invite_list`

Lists all Invites for the current site with their live window state (accepted of quota, next boundary) and waitlist size.

- **Write category:** read-only
- **Access:** requires the `INVITE_READ` permission

No parameters.

`omumu_invite_update`

Updates an Invite's name, quota, schedule (cadence, opening moment, zone), or active flag. Changing cadence, openingDayOfWeek/openingDayOfMonth, openingTime, or zone recomputes the next window boundary. Raising or lowering quota takes effect immediately in the current window without invalidating existing acceptances. Deactivating the Invite (active=false) lifts any deletion guards on its form and sequences. Cadence values: HOUR, DAY, WEEK, MONTH. Day-of-week values: MONDAY..SUNDAY. openingTime format: HH:MM (24h). zone: IANA timezone id (e.g. 'Europe/Oslo').

- **Write category:** standard write
- **Access:** requires the `INVITE_WRITE` permission

Parameter	Type	Required	Description
`active`	boolean	no	Whether the Invite is active; deactivating lifts deletion guards on its form and sequences
`cadence`	string	no	Reset cadence: HOUR, DAY, WEEK, or MONTH
`inviteId`	string	yes	The Invite ID
`name`	string	no	Human-readable name for the Invite
`openingDayOfMonth`	number	no	Day of month for MONTH cadence (1-31)
`openingDayOfWeek`	string	no	Day of week for WEEK cadence (e.g. MONDAY)
`openingTime`	string	no	Opening time in HH:MM for DAY/WEEK/MONTH cadences
`quota`	number	no	Slots available per window
`zone`	string	no	IANA timezone id (e.g. 'Europe/Oslo')

`omumu_optinform_add_field`

Adds a new field to collect user information

- **Write category:** standard write
- **Access:** requires the `OPTINFORM_WRITE` permission

Parameter	Type	Required	Description
`fieldLabel`	string	no	Display label (e.g., 'Email Address')
`fieldName`	string	yes	Internal field name (e.g., 'email', 'firstName')
`fieldType`	string	no	Field type: EMAIL, TEXT, PHONE, or HIDDEN
`formId`	string	yes	The form ID
`placeholder`	string	no	Placeholder text for the input
`required`	boolean	no	Whether the field is required
`sortOrder`	integer	no	Position in the form (0 = first)

`omumu_optinform_create`

Creates a new opt-in form for collecting leads. The form comes with default email and name fields. After creating, use omumu_optinform_update to connect it to an email sequence (emailSequenceId) and/or course (courseId). To embed the form in a page, add `<div data-optin-form="FORM_ID"></div>` to the page content. Forms can also be linked to quizzes as pre-quiz or post-quiz email capture.

- **Write category:** standard write
- **Access:** requires the `OPTINFORM_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	Description or subtitle. Supports rich text when descriptionEncoding is QUILL_JSON.
`name`	string	yes	Internal name for the form (admin use)
`title`	string	no	Display title shown to users

`omumu_optinform_delete`

Deletes an opt-in form and all its submissions

- **Write category:** standard write

- **Access:** requires the `OPTINFORM_WRITE` permission

Parameter	Type	Required	Description
`formId`	string	yes	The form ID to delete

`omumu_optinform_get`

Gets an opt-in form by ID with all its fields. Returns descriptionEncoding (PLAIN_TEXT, QUILL_JSON, or MANAGED_HTML) and trustLine if configured.

- **Write category:** read-only
- **Access:** requires the `OPTINFORM_READ` permission

Parameter	Type	Required	Description
`formId`	string	yes	The form ID

`omumu_optinform_link_sequence`

Connects an opt-in form to an email sequence so submissions trigger subscription

- **Write category:** standard write
- **Access:** requires the `OPTINFORM_WRITE` permission

Parameter	Type	Required	Description
`emailSequenceId`	string	yes	The email sequence ID to link
`formId`	string	yes	The form ID

`omumu_optinform_list`

Lists all opt-in forms for the current site

- **Write category:** read-only
- **Access:** requires the `OPTINFORM_READ` permission

No parameters.

`omumu_optinform_remove_field`

Removes a field from a form by its field ID

- **Write category:** standard write
- **Access:** requires the `OPTINFORM_WRITE` permission

Parameter	Type	Required	Description
`fieldId`	string	yes	The field ID to remove
`formId`	string	yes	The form ID

`omumu_optinform_submissions_list`

Lists submissions for an opt-in form, newest first. Each entry contains the submission ID, timestamp, and the full submitted data (all field values). Returns respondent personal data, so it requires the dedicated leads:read scope (not granted by optinform:read).

- **Write category:** read-only
- **Access:** requires the `LEADS_READ` permission

Parameter	Type	Required	Description
`formId`	string	yes	The form ID
`limit`	number	no	Maximum number of submissions to return (default 50, max 500)

`omumu_optinform_update`

Updates an opt-in form's properties. Use this to connect the form to a course and email sequence, configure the submit button, and set up redirect behavior. Typical funnel setup: create sequence -> create form -> update form with emailSequenceId + courseId + redirectUrl -> embed in page.

- **Write category:** standard write
- **Access:** requires the `OPTINFORM_WRITE` permission

Parameter	Type	Required	Description
`courseId`	string	no	Course to grant access to — creates a user account and enrolls them on form submission
`description`	string	no	Description. Supports rich text when descriptionEncoding is QUILL_JSON.
`descriptionEncoding`	string	no	Encoding for description: PLAIN_TEXT, QUILL_JSON, or MANAGED_HTML
`emailSequenceId`	string	no	Email sequence to subscribe users to — triggers the sequence on form submission
`formId`	string	yes	The form ID
`name`	string	no	Internal name
`redirectUrl`	string	no	URL to redirect after submission
`submitButton`	string	no	Submit button text
`successMessage`	string	no	Message shown after successful submission
`title`	string	no	Display title
`trustLine`	string	no	Small muted text shown below submit button (e.g., 'No spam. Unsubscribe anytime.')

Automations

`omumu_automation_activate`

Activates or deactivates an automation. Active automations will trigger when their conditions are met. Before activating, run automation_simulate for each contact journey that matters and resolve any timing-trap warnings from the save/activate response — activation is the last step of the authoring loop (save, check warnings, simulate, activate).

- **Write category:** standard write

- **Access:** requires the `AUTOMATION_WRITE` permission

Parameter	Type	Required	Description
`active`	boolean	no	true to activate, false to deactivate (default: true)
`automationId`	string	yes	The automation ID

`omumu_automation_create`

Creates a new automation (inactive by default). Recommended authoring loop: 1) automation_save the nodes and connections, 2) read any timing-trap warnings in the save response, 3) automation_simulate the contact journeys that matter (especially 'contact does X, then Y during a delay'), 4) only then automation_activate.

- **Write category:** standard write
- **Access:** requires the `AUTOMATION_WRITE` permission

Parameter	Type	Required	Description
`description`	string	no	Description of what this automation does
`name`	string	yes	The automation name

`omumu_automation_generate_ids`

Generates unique IDs for use when creating new nodes and connections in canvas state

- **Write category:** standard write
- **Access:** requires the `AUTOMATION_WRITE` permission

Parameter	Type	Required	Description
`count`	integer	no	Number of IDs to generate (default: 5, max: 50)

`omumu_automation_get`

Gets an automation with its full canvas state (nodes, connections, configs). Use this to inspect or modify an existing automation.

- **Write category:** read-only
- **Access:** requires the `AUTOMATION_READ` permission

Parameter	Type	Required	Description
`automationId`	string	yes	The automation ID

`omumu_automation_list`

Lists all automations for the current site with their status

- **Write category:** read-only
- **Access:** requires the `AUTOMATION_READ` permission

No parameters.

`omumu_automation_options`

Returns all available sequences, courses (with lessons), tags, forms, offers, quizzes, staff recipients, and supported node types. Use this to find valid IDs when building automation configs. quizzes lists the site's quizzes (id + name) for the quiz_completed trigger. staffRecipients lists who may receive notify_site_staff emails (name + email); defaultRecipient is the authoring user's email. There is no send_email node: to send a single email to a contact, create an email sequence containing one message with zero initial delay and start it with the start_sequence action.

- **Write category:** read-only
- **Access:** requires the `AUTOMATION_READ` permission

No parameters.

`omumu_automation_save`

Saves the full canvas state (nodes + connections) for an automation. The automation must be inactive. Canvas state format: {"nodes": [{"id": "...", "type": "tag_applied", "x": 0, "y": 0, "config": {...}}], "connections": [{"id": "...", "from": "nodeId", "to": "nodeId", "type": "default"}]}. Use automation_generate_ids to get IDs for new nodes/connections. Use automation_options to get valid IDs for sequences, courses, tags, forms, offers, quizzes. The quiz_completed trigger's config takes {"questionSetId": quizId} (see quizzes in automation_options); it fires only for identified respondents (those who left an email) when their quiz result is saved. The notify_site_staff action's config takes {"recipient": email, "subjectPrefix": optional} — the recipient MUST be a Headmaster or Staff member on the site (see staffRecipients in automation_options; defaultRecipient is the authoring user); other addresses are rejected. The submission_received / submission_rated / submission_reviewed triggers' config takes {"assignmentId": ...} (see

courses[].assignments in automation_options) — a submission trigger without an assignmentId never fires. The unlock_module action's config takes {"courseId": …, "moduleId": …} (see courses[].modules in automation_options) and appends a MODULE-scoped access entry for the contact — meaningful on a course whose gating mode is SEQUENTIAL; both ids must belong together on this site or the save is rejected. The module_completed trigger's config takes {"courseId": …, "moduleId": …} (see courses[].modules in automation_options) and fires when the contact's completion of exactly that module is recorded; both ids must belong together on this site or the save is rejected. The course_completed trigger's config takes {"courseId": …} and fires when the contact's completion of that course is recorded. Completion triggers are side-effects only (celebrate, tag, follow up) — on a SEQUENTIAL course the next module unlocks by itself, so do not wire module_completed to unlock_module for gating. Pair submission_received with unlock_module to open the next module when a student posts their work. Pair quiz_completed (or form_submit) with notify_site_staff to email a staff member a digest of every answer, the bucket, and the score. There is no send_email node: to send a single email to a contact, create an email sequence containing one message with zero initial delay (omumu_email_sequence_create, initialDelay PT0S) and start it with the start_sequence action. The response includes timing-trap warnings (e.g. stop-sequence-before-delayed-start: a STOP_SEQUENCE may fire before a WAIT-delayed START_SEQUENCE and silently no-op; fix by starting the sequence immediately and moving the delay into the sequence's initialDelay). Warnings are non-blocking, but do not activate past one without first proving the funnel correct with automation_simulate.

- **Write category:** standard write
- **Access:** requires the `AUTOMATION_WRITE` permission

Parameter	Type	Required	Description
`automationId`	string	yes	The automation ID
`canvasState`	object	yes	The full canvas state JSON with nodes and connections

`omumu_automation_simulate`

Runs a scenario against the site's automations in memory (no emails sent, nothing written) and returns the effect timeline plus any timing-trap warnings. All of the site's automations run together so cross-flow races (e.g. a

purchase stopping a delayed abandonment sequence) are visible. ‘events’ is an array of {“at”: duration, “trigger”: triggerType, “contact”: alias}: ‘at’ is a duration from t0 as ISO-8601 (PT5M) or a friendly form (0s, 5m, 1h, 2d, 1w); ‘trigger’ is a contact trigger type (form_submit, purchase, tag_applied, page_visit); ‘contact’ is an optional alias (default ‘contact’) so several events can describe one contact’s journey. Each timeline entry is {atSeconds, at, contact, kind, target, effective}: kind is APPLY_TAG, REMOVE_TAG, START_SEQUENCE, STOP_SEQUENCE, EMAIL_SENT, ENROLL_COURSE, UNLOCK_MODULE, GRANT_ROLE or NODE_REJECTED; ‘effective’ is false for a no-op (e.g. a stop with no live subscription). Use this to verify a funnel behaves before activating it: don’t just look for the expected effects — scan the timeline for effective=false on stops/removals and for EMAIL_SENT entries the contact should NOT receive (an ineffective STOP_SEQUENCE followed by a later EMAIL_SENT is the classic cart-abandonment race). Simulating is step 3 of the authoring loop: save, check warnings, simulate the journeys that matter, then activate.

- **Write category:** read-only
- **Access:** requires the `AUTOMATION_READ` permission

Parameter	Type	Required	Description
`automationId`	string	yes	The automation ID to simulate (its site’s automations all run together)
`events`	array	yes	Ordered scenario events: [{“at”: “5m”, “trigger”: “purchase”, “contact”: “buyer”}]

Ad Validation & Offer DNA

`omumu_offer_dna_architect_brief`

Returns the complete offer-architecture task for a DNA — the method rules, the embedded research verbatim, interview context, the format-intent constraint when declared, the freshly-resolved fenced existing material, and the exact output schema. Pure read: no AI call, no budget interaction. Run your own architecture using the instructions and context, then submit with omumu_offer_dna_architect_submit. Step order: omumu_offer_dna_research_submit → omumu_offer_dna_architect_brief → omumu_offer_dna_architect_submit → omumu_offer_dna_launch.

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

Parameter	Type	Required	Description
`id`	string	yes	The Offer DNA id

`omumu_offer_dna_architect_submit`

Validates and stores an offer architecture the calling agent produced itself — no AI budget is spent. The proposal object must have architecture, compression, and score matching the schema returned by omumu_offer_dna_architect_brief. Degenerate, empty (no solutions and no stack), or malformed submissions are rejected with a descriptive error; nothing is stored on failure. Re-submitting replaces all three prior sections. Next step: omumu_offer_dna_launch.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`id`	string	yes	The Offer DNA id
`proposal`	object	yes	The proposal object (architecture, compression, score) matching the schema from omumu_offer_dna_architect_brief

`omumu_offer_dna_create`

Starts a research-based offer document (ADR-0034) from a creator interview. Returns its id; follow with omumu_offer_dna_research_brief.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`audience`	string	yes	Who it is for — the more specific the better
`container`	string	no	The intended size of the thing (format intent, not a price bucket): TINY (a single video or one short lesson), NORMAL (a few lessons up to modest modules), FLAGSHIP (a multi-module program), or UNDECIDED (the architect infers the scale — the default)
`idea`	string	yes	What the creator wants to sell, in a sentence or two
`knownPains`	string	no	Pains the creator already suspects (the research verifies or challenges them)
`priceIdea`	string	no	The creator's price idea, if any
`scopeNotes`	string	no	Free-text medium/cadence/delivery notes, e.g. "one 45-minute screencast" or "6 modules over 6 weeks"
`seedCourseId`	string	no	An existing Course on this site to build on — the architect reads its module and lesson titles as the real deliverables
`seedOutline`	string	no	Pasted outline or lesson material drafted outside Omumu — the offer is architected around it instead of an invented curriculum

`omumu_offer_dna_get`

Returns the full Offer DNA document: interview, cited research, architecture, compression (touchstone + big four), and score.

- **Write category:** read-only

- **Access:** requires the `OFFER_READ` permission

Parameter	Type	Required	Description
`id`	string	yes	The Offer DNA id

`omumu_offer_dna_launch`

The lab's final step: creates the Offer, authors its validation page from the DNA (one AI run, billed to the daily budget; falls back to starter copy when the model is unavailable), scaffolds the funnel, and links the DNA to the offer. A course-shaped DNA (container TINY/NORMAL/FLAGSHIP) attaches a real-but-empty Course and sets accessOpensAt so presale buyers hold an entitlement that later fills up. Requires omumu_offer_dna_architect_submit to have completed first.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`accessOpensAt`	string	no	When a course-shaped offer's content opens, ISO date or date-time; defaults to 30 days from launch
`currency`	string	no	USD, EUR, NOK or SEK; defaults to USD
`id`	string	yes	The Offer DNA id
`offerName`	string	yes	The offer's final name (the human's choice among the candidates, or a new one)
`totalAmountCents`	number	yes	Total price in cents (e.g., 2900 for 29.00)
`touchstone`	string	no	The chosen touchstone; defaults to the DNA's recommendation

`omumu_offer_dna_list`

Lists the site's Offer DNA documents, newest first, with their completion state.

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

No parameters.

`omumu_offer_dna_research_brief`

Returns the complete voice-of-customer research task for a DNA — the method rules, interview context, the exact output schema the submission must match, and an Offer Lab fallback pointer. Pure read: no AI call, no budget interaction. Run your own web research using the instructions and context, then submit with omumu_offer_dna_research_submit. If you lack web-research capability, direct the creator to the Offer Lab instead. Next step after submitting: omumu_offer_dna_architect_brief.

- **Write category:** read-only
- **Access:** requires the `OFFER_READ` permission

Parameter	Type	Required	Description
`id`	string	yes	The Offer DNA id

`omumu_offer_dna_research_submit`

Validates and stores research the calling agent ran itself — no AI budget is spent. The research object must match the schema returned by omumu_offer_dna_research_brief. Degenerate, empty-findings, or malformed submissions are rejected with a descriptive error; nothing is stored on failure. Re-submitting replaces the research section while leaving any existing architecture untouched. Next step: omumu_offer_dna_architect_brief.

- **Write category:** standard write
- **Access:** requires the `OFFER_WRITE` permission

Parameter	Type	Required	Description
`id`	string	yes	The Offer DNA id
`research`	object	yes	The research object matching the schema from omumu_offer_dna_research_brief

Podcasting

`omumu_podcast_episode_create`

Creates a draft episode on the site's show from an already-uploaded audio resource (upload the file first via the multipart resource upload endpoint; mp3, m4a, and wav are accepted). Mastering starts automatically and the episode is returned in PROCESSING status; poll omumu_podcast_episode_get until it reaches READY, then publish.

- **Write category:** standard write
- **Access:** requires the `PODCAST_WRITE` permission

Parameter	Type	Required	Description
`audioResourceId`	string	yes	ID of the uploaded audio resource to master into this episode
`episodeNumber`	number	no	Optional episode number within the season
`episodeType`	string	no	Episode type: 'full' (default), 'trailer', or 'bonus'
`explicit`	boolean	no	Per-episode explicit override; omit to inherit the show's flag
`season`	number	no	Optional season number
`showNotes`	string	no	Show notes describing the episode
`title`	string	yes	The episode title

`omumu_podcast_episode_get`

Gets one episode by id with its processing status, duration, error message when failed, and pubDate when published.

- **Write category:** read-only
- **Access:** requires the `PODCAST_READ` permission

Parameter	Type	Required	Description
`episodeId`	string	yes	The episode id

`omumu_podcast_episode_list`

Lists the show's episodes with their processing status (PROCESSING, READY, FAILED, PUBLISHED), duration, error message when failed, and pubDate when published.

- **Write category:** read-only
- **Access:** requires the `PODCAST_READ` permission

No parameters.

`omumu_podcast_episode_publish`

Publishes a READY episode: the pubDate is stamped now and the episode appears in the RSS feed. A PROCESSING or FAILED episode is refused with its current status.

- **Write category:** standard write
- **Access:** requires the `PODCAST_WRITE` permission

Parameter	Type	Required	Description
`episodeId`	string	yes	The episode id

`omumu_podcast_episode_retry`

Re-runs mastering on a FAILED episode, returning it to PROCESSING. Episodes in any other status are refused with their current status.

- **Write category:** standard write
- **Access:** requires the `PODCAST_WRITE` permission

Parameter	Type	Required	Description
`episodeId`	string	yes	The episode id

`omumu_podcast_show_get`

Gets the site's podcast show: title, slug, description, language, and the public RSS feed URL. Errors if the site has no show yet (shows are created in the podcast admin).

- **Write category:** read-only
- **Access:** requires the `PODCAST_READ` permission

No parameters.

Product Skills

`omumu_skill_get`

Returns the complete SKILL.md instructions for a skill the user is entitled to. Use after omumu_skills_list to load the workflow and follow its steps. Returns an error if the user is not entitled or the skill is not published.

- **Write category:** read-only
- **Access:** requires the `SKILL_READ` permission

Parameter	Type	Required	Description
`slug`	string	yes	The skill slug, e.g. 'create-course'

`omumu_skills_list`

Lists every published Omumu skill (workflow for AI assistants) visible to the caller's site, marking which the user is entitled to read. Use to discover what workflows are available — course creation, validation, sales pages, etc. Set entitledOnly=true to hide skills the user has not purchased.

- **Write category:** read-only
- **Access:** requires the `SKILL_READ` permission

Parameter	Type	Required	Description
`entitledOnly`	boolean	no	If true, only return skills the user is entitled to. Default false.

Permission scopes

Permission	Grants
ANALYTICS_DETAILED	Access detailed analytics and conversion data
ANALYTICS_READ	Read analytics and performance data
API_KEY_MANAGE	Manage API keys for the site
AUTOMATION_READ	Read automations, canvas state, and options
AUTOMATION_WRITE	Create, update, activate, and deactivate automations
COURSE_DELETE	Delete courses and course components
COURSE_READ	Read courses and course structure
COURSE_WRITE	Create and update courses, modules, and lessons
EMAIL_INBOX_READ	Read received emails in the inbox
EMAIL_SEQUENCE_READ	Read email sequences and their content
EMAIL_SEQUENCE_WRITE	Create and update email sequences and follow-ups
FUNNEL_READ	Read funnels and their branched step graph
FUNNEL_WRITE	Add upsell steps to funnels (requires UPSELLS feature on the site's tier)
INVITE_READ	Read Invites, window state, and waitlist size
INVITE_WRITE	Update Invites (name, quota, schedule, active flag) and launch new Invites, which creates a course, email sequences, an opt-in form, and a page
LEADS_READ	Read form submissions including respondent personal data
MEDIA_GENERATE	Generate images using AI
OFFER_DELETE	Delete offers
OFFER_PUBLISH	Publish and manage offer lifecy- [illegible]
[illegible]	[illegible]