ServiceMinder API Guidelines

07/17/2026 11:49 am CDT

Overview

This guide outlines the recommended best practices for integrating with the ServiceMinder API. Following these guidelines will help you build secure, reliable, and scalable integrations while minimizing unnecessary API traffic.

Whether you're syncing data, processing events, building dashboards, or developing custom applications, these recommendations will help ensure your integration performs efficiently and follows ServiceMinder's supported patterns.

This article will review:

Do's & Don'ts

Do:

- **Subscribe to drip trigger events** for change detection.
  - 70+ event types are available — contact added/updated, appointment scheduled/completed/canceled, proposal created/accepted/declined, invoice paid, payment received, and more. API-originated writes fire dedicated events (`ContactAddedApi` , `ContactUpdatedApi` , `AppointmentScheduledApi`), so you can react to your own integration's activity.
- **Use** `UpdatedFrom` / `UpdatedThrough` (and `CreatedFrom` / `CreatedThrough` where available) on `Query` endpoints to fetch only what changed since your last sync. Never re-pull the full table on a schedule.
- **Paginate correctly.**
  - `Query` endpoints use `Skip` / `Take` with a maximum page size of **500** (Contacts `Locate` uses `Skip` / `Limit` , default 100, max 500). Results are ordered by `Id` ascending and the total record count is returned, so you can page deterministically. Page sequentially — don't fire all pages in parallel.
- **Reference records by ID, not name.**
  - Prefer `ServiceId` over `ServiceName` — ambiguous names return an explicit failure rather than guessing.
- **Call the API server-side.**
  - The API key authenticates the request and must be kept secret. CORS is open, so browser calls technically work, but that exposes your key — always call from your backend.
- **Check** `ResultCode` . `/api` responses return an integer `ResultCode` : `0` = success, `1` = failure, with a `Message` .
  - Key your success/error handling off the integer.
- **Handle failures with backoff.**
  - On any error or timeout, back off exponentially and retry later — never retry immediately in a tight

loop.

- **Treat webhook delivery as fire-and-forget.**
  - Return HTTP 200 quickly from your webhook receiver; do heavy processing asynchronously. Non-200 responses are logged but **not retried**. If you need guaranteed delivery, use the event-queue mode instead of webhooks.
- **Let bulk downloads finish before starting another.**
  - Downloads are concurrency-limited per org and enqueue is idempotent — an identical in-flight request returns the existing download. Poll download status for completion rather than re-submitting.

Don't

- **Don't poll on a fixed short interval.**
  - Once-per-second (or anything close) for a single org is far more than needed.
- **Don't burst.**
  - Firing hundreds of calls in a few seconds concentrates load and can degrade the whole platform. Space requests out; process serially or with tight concurrency limits.
- **Don't re-download or re-query full datasets on a schedule.**
  - Full loads are a one-time (or rare) operation via bulk download; steady state is push events + incremental queries.
- **Don't fetch the same record repeatedly.**
  - Cache what you've already retrieved and rely on change events to know when to refresh it.
- **Don't request page sizes above 500.**
  - They're silently clamped to 500, so a "bigger page" just means you're not paging.
- **Don't embed the API key** in client-side / browser / mobile code.
- **Don't rely on webhook retries.**
  - There are none. Use the event queue if you can't tolerate a missed delivery.
- **Don't assume** `datasubscriber/register` / `list` **exist.**
  - Webhook and event-subscription setup is done via Drip Rules configured in the ServiceMinder app, and DataSubscriber credentials are provisioned by ServiceMinder — there is no self-service webhook-registration endpoint.

Downloading Data

For anything larger than a page or two, use the**DataSubscriber bulk download** instead of paging through `Query` endpoints. It's an asynchronous CSV export you kick off, poll, and then retrieve — three steps:

1. **Start** — `POST /api/datasubscriber/startdownload` with the record kind and any filters. Returns a `download_id`. (An identical, still-in-flight request returns the existing `download_id` with `duplicate: true` rather than starting a second — enqueue is idempotent, and downloads are concurrency-limited per org.)

2. **Poll status** — `POST /api/datasubscriber/downloadstatus` with the `download_id` until it reports complete. Poll on a reasonable interval (e.g. every several seconds), not in a tight loop.

3. **Retrieve** — `POST /api/datasubscriber/getdownload` with the `download_id` to fetch the CSV once complete.

Record kinds cover the main entities, including: `contacts`, `contact-notes`, `appointments`, `appointment-lines`, `proposals`, `invoices`, `invoice-lines`, `payments`, `deposits`, `parts`, `services`, `service-agents`, `organizations`, `users`, `channels-campaigns`, `accounting-classes`, and drip/campaign/forecast data.

Filters — every kind supports `CreatedFrom` / `CreatedThrough`, `UpdatedFrom` / `UpdatedThrough`, `OrganizationId`,

and `RowId`. Use the date filters for incremental exports (only what changed since your last run) rather than re-exporting everything.

Auth note: DataSubscriber access uses a brand-level credential provisioned by ServiceMinder, which is separate from the per-org API key used by the rest of `/api`. Request it from ServiceMinder if you need bulk export.

Pattern: seed your integration with a one-time full bulk download, then keep it current with change events (below) and periodic incremental downloads — never a scheduled full re-export.

Safely Building Dashboards, Custom Tools, & Integrations

> **Golden rule: don't call the ServiceMinder API on every page load, widget render, or dashboard refresh.** That pattern turns one popular dashboard into thousands of redundant calls.

Instead, **sync ServiceMinder data into your own datastore and build on top of that:**

1. **Seed** your datastore with a one-time bulk download (above).
2. **Keep it current** with change events — a drip-trigger webhook (`JsonPost`) or the event queue (`Fetch` / `Clear`) — so your copy updates when records change, with no polling.
3. **Reconcile** occasionally with `Query` + `UpdatedFrom` / `UpdatedThrough` on a slow cadence (e.g. nightly) to catch anything missed.
4. **Render** dashboards, reports, and custom tools entirely from your local copy.

Why this is the safe pattern:

- **Fast for your users** — queries hit your store, not a remote API.
- **No load on ServiceMinder** that scales with how many people view your dashboard.
- **Resilient** — your tool keeps working during brief API slowdowns or maintenance.
- **Near-real-time** — event-driven updates keep your copy fresh without a polling loop.

If you genuinely need live, on-demand reads (e.g. a lookup tool that fetches one contact when a user searches), that's fine — scope the call to the single record the user asked for, call it server-side, and cache the result. The anti-pattern is *background* or *per-refresh* calls that run whether or not anyone is looking.

API Basics

Topic	Detail
Base URL	`https://serviceminder.com/api` (public) · `https://serviceminder.io/api` (partner)
Route shape	`POST /api/{controller}/{action}` — e.g. `POST /api/contacts/locate`, `POST /api/appointments/book`
Auth	API key sent as an `"ApiKey"` field **in the JSON request body** (not a header). Key must be active; requests are rejected for canceled accounts / expired trials.
Success indicator	Integer `ResultCode`: `0` = success, `1` = failure, plus `Message`.
Pagination	`Skip` / `Take`, max page size **500**, ordered by `Id`; total count returned. Contacts `Locate` uses `Skip` / `Limit` (default 100, max 500).

Topic	Detail
Incremental sync	`UpdatedFrom` / `UpdatedThrough` (and `CreatedFrom` / `CreatedThrough` on some endpoints).
Self-documentation	`GET /api/Home/Index` renders a live list of available endpoints.

Quick Decision Guide

You want to…	Use This	Not That
Load all existing records once	DataSubscriber bulk download	Paging through every `Query` endpoint
Know when a contact/appointment/proposal/invoice changes	Drip trigger → webhook or event queue	Polling `Query` on a timer
Catch up after downtime / reconcile	`Query` with `UpdatedFrom` / `UpdatedThrough`, slow cadence	Full re-download
Guaranteed delivery of events	Event queue (`Fetch` / `Clear`)	Webhooks (no retry)
Real-time, low volume	Webhook (`JsonPost`)	—
