Synchro User Guide

Version 2.7.9.055

28 September 2026

Contents

Synchro User Guide

Documented against	**SynchroServer `2.7.9.055`**
SynchroUI	`1.0.134`
SynchroAPI	`v1` (`/api/v1/`)
SynchroAgent	`1.1.5`
Last updated	2026-09-28
Replaces	SynchroServer User Guide v2.4 (2017-04-18)

This guide explains what Synchro is, how it models your data, and how to work with it through the web interface, the REST API and the agents.

It is the starting point rather than the whole story. Three companion documents go deeper on their own subjects, and this guide points to them rather than repeating them:

Document	Covers
SynchroUI User Guide	every control in the web interface
SynchroAPI Reference	every endpoint, with request and response shapes
SynchroAgent User Guide	installing and running the host agent

Problems with this documentation, or with Synchro itself, should go to support@affatechnology.com.

1. Introduction

1.1 What is Synchro?

Synchro is a real-time enterprise infrastructure-monitoring tool. It acquires and manages data and events from many sources across an enterprise, then collects, aggregates, historises, alerts, visualises and reports on them.

Think of Synchro as a spreadsheet whose field values can change every second, or more often if required. Like a spreadsheet, Synchro does not need to understand the meaning of the data it processes.

A field in Synchro, together with the values it takes over time, is a **time series**. A typical Synchro installation processes hundreds of thousands of time series updates every second.

Three properties characterise it:

Efficient. A single Synchro infrastructure sustains more than 200,000 data updates per second on commodity server hardware.

Granular. All data is kept at second-level precision by default, and can be historised at that precision for years where it is needed.

Insightful. Masses of statistics are recalculated every second, so the data that needs attention surfaces on its own rather than waiting to be queried.

1.2 What Synchro does with your data

For each time series, Synchro maintains two kinds of statistics, both continuously rather than on demand.

ByTimeStatistics

For every second of the day, within every time series, Synchro keeps the latest value, minimum, maximum, average, count and sum of everything received in that second.

Field	Meaning
`LAST`	the most recent value received in that second
`MIN`	smallest value received
`MAX`	largest value received
`AVG`	mean of the values received
`COUNT`	how many values were received
`SUM`	total of the values received

A second in which nothing arrived has no statistics, which is itself information: it is how Synchro can tell you that something stopped reporting.

PeriodStatistics

A **period** is a continuous interval within the day. For each period and each time series, Synchro keeps the same six figures plus two more:

Field	Meaning
`TIME`	time of the last update
`DTIME`	time elapsed since the last update

`DTIME` deserves particular attention. It is how you detect silence: a series whose `DTIME` keeps growing is one whose source has stopped, and no threshold on the value itself would reveal that.

Hourly, morning, evening, work-hours and daily periods are defined by default. You may define as many more as you need, and they may overlap.

1.3 How a time series is identified

Synchro identifies every time series with four names:

```
COMPONENT · TYPE · METRIC · TARGET
```

By convention:

- **Type** and **Metric** describe *what is being measured*
- **Component** and **Target** describe *what it is measured on*

A processor measurement from a host called `web01` is therefore:

Identifier	Value
Component	`web01`
Type	`HOST`
Metric	`CPU`
Target	`HOST`

and the memory used by a process named `postgres` on that same host is:

Identifier	Value
Component	`web01`
Type	`PROC`
Metric	`RSS`
Target	`postgres`

The component usually identifies the machine or application sending the data, and is supplied once when a source connects rather than repeated on every measurement. The other three travel with each value.

This four-part scheme is what makes wildcard selection useful: asking for `*·PROC·RSS·postgres` gives you the memory of that process across every machine reporting it, with no configuration to maintain as machines come and go.

1.4 Selecting by value, not only by name

One of Synchro's more distinctive features is that a time series can be selected by the *value* of its PeriodStatistics as well as by its name.

That makes a real-time list of the top N clients, products or systems — updated every second — a matter of asking for it, rather than something that has to be computed and maintained. The same mechanism supports comparing what is missing or newly present between two dates or two periods of the day, which turns "what changed on these machines overnight?" into a single query.

1.5 Derived series

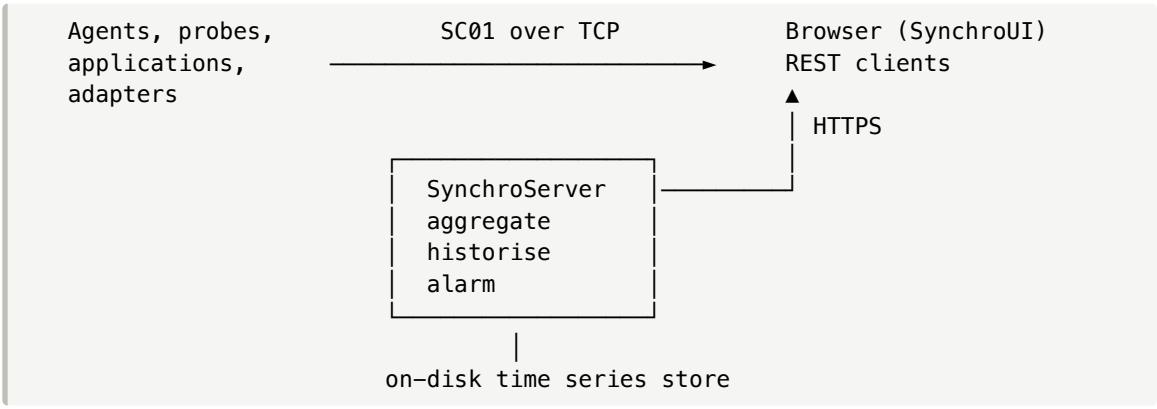
Synchro can also create time series from your data rather than from a source:

- **Aggregates** group related series together, or combine days of history

- **Moving functions** such as moving averages (MAVG) produce a smoothed series from a raw one
- **Alarms** are themselves time series, so an alarm's own history can be charted, aggregated and alarmed on like any other data

Because derived series are ordinary time series, everything in this guide applies to them equally.

1.6 Architecture overview



SynchroServer is a multi-threaded Java application and the heart of the installation. It receives data from agents, probes, applications and adapters, then aggregates, historises, reports and raises alarms as needed.

Data arrives over a deliberately trivial text-based TCP protocol, described in section 4. That simplicity is a design decision: writing an adapter for a new data source is a small job rather than a project.

SynchroUI is the browser interface. It talks to SynchroServer's embedded web server over HTTP, through the same REST API available to your own code.

SynchroAgent reports a host's own processor, memory, disk, filesystem, network and per-process activity. It is a single static binary with no runtime to install, available for Linux, Windows and macOS.

1.7 What changed since version 2.4

Version 2.4 of this guide described Synchro as it stood in 2017. Readers familiar with it should note the following.

Then	Now
SynchroWeb, a Java applet interface	**SynchroUI**, a React application in any modern browser
Configuration through the admin interface	**REST API** and SynchroUI; the legacy interface is disabled by default
SynchroClient for Solaris, Linux, Windows	**SynchroAgent** for Linux, Windows and macOS, plus the existing Java client
Passwords in a simple store	Hashed storage, password policy and login rate limiting
—	**Scheduled event monitoring**
—	Views can be **shared by link**

The concepts in sections 1.2 to 1.5 are unchanged. Time series, periods, ByTimeStatistics and PeriodStatistics behave exactly as they always have, and data recorded by earlier versions remains readable.

2. Using SynchroUI

SynchroUI is the browser interface to SynchroServer. This section describes what it is for and how it is organised; the SynchroUI User Guide documents every control.

2.1 Getting in

Open SynchroUI at your server's address and sign in. A freshly installed system has a default account whose password must be changed on first use; see Password Management.

A view can also be opened without signing in, if it has been shared. See section 2.5.

2.2 How the screen is organised

Area	Purpose
Header bar	version, refresh control, downloads, alarm bell, views menu, current view name, account
Alarm status strip	current alarm state at a glance, always visible
Variables bar	values substituted into the panels below
Panels	the dashboard itself

2.3 Panels

A panel is one window onto your data. Panels are resized and reordered directly, and a view may hold as many as are useful.

Panel type	Shows
Search	find time series by name
Time Series Graph	values over time
Heatmap	many series at once, by intensity
Histogram	distribution of values
Statistics	the ByTime and Period figures as a table
Alarms	current alarm state
Data	raw values
H-Bar / V-Bar Graph	comparison across series
Card	a single value, large

Each panel carries its own time range, or follows the view's.

2.4 Time range and refresh

Every panel is a window onto a time range. Because Synchro keeps data at one-second precision, a range may be as narrow as a few seconds or as wide as years, and the interface fetches an appropriate resolution for the span you ask for.

The refresh control governs how often panels re-query. Live dashboards typically refresh every second; historical analysis need not refresh at all.

2.5 Views: save, load, share

A **view** is a saved dashboard: its panels, their arrangement, their time ranges and its variables.

Views can be saved under your own account, loaded again later, deleted, and **shared**. Sharing produces a link that opens the view directly — which is what makes a dashboard usable by people who do not otherwise use Synchro, such as a wall display or a colleague who only needs one number.

Shared links open in **guest view mode**: the view is visible, but nothing can be changed or saved.

2.6 Variables

Variables let one view serve many subjects. A view built around a variable for the component name can be pointed at any host without being rebuilt, and the value travels in the URL, so a link can carry it.

2.7 Alarms

The alarms manager lists alarm rules, organised into groups, and lets them be created, edited, enabled and disabled. Alarm state appears in the status strip and the bell, and alarm history is itself a time series. Alarms are covered in full in section 5.

2.8 Events

The events tab covers scheduled event monitoring: work that is expected to happen by a certain time, where the absence of an event matters as much as its content.

3. Using the API

Everything SynchroUI does, it does through the REST API, and that API is available to your own code. The full reference is the SynchroAPI Reference; what follows is a map.

All endpoints live under `/api/v1/`.

3.1 Authentication

Sign in with `POST /auth/login` and use the returned session for subsequent calls. `GET /auth/me` reports who you are, and `POST /auth/password` changes your own password.

A subset of endpoints under `/public/` needs no authentication at all, which is how shared views and guest mode work.

3.2 What is available

Group	Endpoints	For
Health	`GET /health`	monitoring the monitor
Index	`GET /index`, `GET /public/index`	discovering which time series exist
Data	`GET /data/query`, `GET /data/timeseries`	reading values, and series for charting
Periods	`GET /periods`	the periods defined on the server

Group	Endpoints	For
Views	`GET/POST/DELETE /views/{user}/{name}`	saving and loading dashboards
Alarms	`GET /alarms/state`	current alarm state
Events	`GET /events`, `POST /events/{id}/start`, `/complete`	scheduled event monitoring
Layouts	`GET/POST/DELETE /layouts/{id}`	dashboard layouts
Downloads	`GET /download/{filename}`	release artefacts

Administrative endpoints sit under `/admin/` and require the appropriate access level:

Group	Endpoints
Alarms	`GET/POST /admin/alarms`, `PUT /admin/alarms/enabled`, `POST /admin/alarms/reload`, `PUT/DELETE /admin/alarms/file/{name}`
Users	`GET/POST /admin/users`, `PUT/DELETE /admin/users/{uid}`, `POST /admin/users/{uid}/password`
Configuration	`GET/POST /admin/config/{name}`
Jobs	`POST /admin/jobs/{name}`
Releases	`GET /admin/releases`, `GET/POST/DELETE /admin/downloads`

The per-file alarm endpoints (`/admin/alarms/file/{name}`) were added in 2.7.9.055.

3.3 Cross-origin requests

The API supports CORS, so a dashboard or tool served from elsewhere can call it directly from the browser. Configuration is covered in the API reference.

4. Getting data in

Synchro accepts data from anything that can open a TCP socket and write a line of text. That is the whole integration surface.

4.1 The SC01 protocol

A source connects to the server's stat port — **9123** by default — and exchanges tab-separated lines.

1. Identify. The source announces the component name its data belongs to:

```
SC01 <TAB> init <TAB> web01
```

2. Receive configuration. The server replies with the sampling interval it wants and whether it would like a configuration dump:

```
SS01 <TAB> conf <TAB> Stat=1 <TAB> DumpConf=0
```

3. Send measurements. Thereafter the source sends one line per sample: a timestamp followed by any number of `key=value` pairs.

```
SC01 <TAB> stat <TAB> 20260928.143001 <TAB> HOST_CPU_HOST=12.50 <TAB>
HOST_MEM_HOST=41.20
```

Each key is `TYPE_METRIC_TARGET`, split on underscores into exactly three parts. The component comes from the `init` line, so it is not repeated. Values are decimal.

4. Close. `SC01 <TAB> close` ends the session cleanly.

Two rules matter when generating keys:

- A key must contain **exactly two underscores**. If a type, metric or target contains one of its own, the split breaks. Replace it with a hyphen.
- Characters outside `a–z A–Z 0–9 @ . _ = : ; - ~` are replaced by the server, so a name containing spaces or slashes will not arrive as written.

Sending for another component

The `mstat` verb takes a four-part key with the component first, which lets one connection carry data for several components:

```
SC01 <TAB> mstat <TAB> 20260928.143001 <TAB> web01_HOST_CPU_HOST=12.50
```

This is how an adapter can report on machines other than the one it runs on.

4.2 Timestamps

Timestamps are `YYYYMMDD.HHMMSS` in the **sending machine's local time**. A machine whose clock or time zone differs from the server will have its data filed under the wrong time, and the server rejects samples dated outside its read-write window. Keep clocks synchronised and time zones consistent.

4.3 SynchroAgent

For host metrics there is no need to write anything: SynchroAgent reports processor, per-core, memory, disk, filesystem, network and per-process activity out of the box, on Linux, Windows and macOS.

It is a single static binary. Installation, configuration and the full list of metrics it reports are in the SynchroAgent User Guide.

4.4 Writing your own adapter

Anything that can write a line to a socket can feed Synchro. In practice an adapter is a loop that connects, sends `init`, then sends a `stat` line per interval, and reconnects if the connection drops.

Two behaviours are worth copying from SynchroAgent:

Never give up. Retry the connection indefinitely rather than exiting. A server restart should not require restarting every adapter.

Do not buffer during an outage. Sending a backlog when the connection returns produces a burst of back-dated samples that the server will reject anyway. A gap in the data is the honest representation of a gap in collection.

5. Alarms

An alarm is a rule evaluated continuously against your time series. When a value leaves the band you called normal, the alarm changes severity, and that change is recorded, displayed and — because an alarm is itself a time series — kept as history you can chart and alarm on in turn.

5.1 Severity

An alarm has four severities, defined as value bands:

Severity	Band
OK	`OKMin` to `OKMax`
Warning	`WarnMin` to `WarnMax`
Critical	`CriticalMin` to `CriticalMax`
Fatal	`FatalMin` to `FatalMax`

Bands are explicit rather than derived, so they need not be contiguous and need not be ascending. A metric that is wrong when it falls — a queue that should never empty, a throughput that should never drop — is expressed by ordering the bands the other way.

`Raise level` sets the minimum severity a rule will report, which is how a rule can track a value without generating traffic until it matters.

`Raise criteria` — **Both**, **Rise** or **Fall** — decides whether crossing into a band counts when the value is rising, falling, or either.

5.2 Choosing which series a rule watches

A rule does not name a single time series. It carries a regular expression for each of the four identifiers, and applies to everything that matches:

Filter	Default	Example
Component	`.*`	`^web[0-9]+$`
Type	`.*`	`HOST`
Metric	`.*`	`CPU`
Target	`.*`	`HOST`
Date	`.*`	`D-00` (today)
Period	`.*`	`00@24` (all day)

This is what makes alarms maintainable across a changing estate. A rule matching `Component .*`, `Type HOST`, `Metric CPU` covers every machine that reports processor use, including ones installed tomorrow, with nothing to update when they arrive.

5.3 Choosing which statistic to test

Because Synchro keeps several statistics per series, a rule states which one it evaluates:

Function	Tests
`Last`	the most recent value

Function	Tests
`Average`	the mean over the period
`Min` / `Max`	the extremes over the period
`Sum`	the total over the period
`Count`	how many values arrived
`DTime`	**seconds since the last update**

`DTime` deserves singling out. Alarming on a *value* only works while data is arriving; a source that dies stops sending and its last value sits there looking healthy forever. A `DTime` rule catches silence, and it is usually the most important rule on any component. If you write one alarm per data source, write that one.

`Count` serves a related purpose: a series still reporting, but reporting far fewer samples than it should.

5.4 Keeping alarms quiet enough to be useful

An alarm that fires constantly is ignored, which makes it worse than no alarm. Four fields control that.

Field	Effect
Delay (s)	stay breached this long before raising
Snooze (s)	after raising, suppress new triggers for this long
Repeat (s)	after the snooze, re-raise every N seconds while still breached
Exemption	ignore this many breaches before counting at all

`Delay` is the one to reach for first. At one-second resolution almost everything crosses a threshold occasionally; a delay of thirty or sixty seconds removes that noise without hiding a real fault.

`UseByTime` makes a rule respect period exemption windows, so a nightly batch that legitimately saturates a disk need not page anyone.

5.5 Where rules live

Rules are stored on the server as JSON, grouped into named files under `conf/alarm-json.d/`. Each file holds one or more named groups, and each group holds its rules:

```
{
  "Alarms": [
    {
      "Name": "DEFAULT",
      "Active": true,
      "AlarmsItems": [
        {
          "Description": "DefaultAlarm",
          "Component": "Synchro",
          "Type": "SYNCHRO$",
          "Metric": "MEM-USED-MB",
          "Target": "SERVER",
          "Date": "D-00",
          "Period": "00@24",
          "Function": "LAST",
          "OKMin": 0,    "OKMax": 1000,
          "WarnMin": 1000, "WarnMax": 2000,
          "CriticalMin": 2000, "CriticalMax": 3000,
          "FatalMin": 3000, "FatalMax": 999999
        }
      ]
    }
  ]
}
```

Grouping matters operationally: a whole group can be switched off with `Active`, which is how you silence a subsystem during planned work without editing individual rules.

Three ways to manage them:

- **SynchroUI** — the alarms manager, with a form per rule. The usual choice.
- **The API** — `GET/POST /admin/alarms` for the whole set, `PUT/DELETE /admin/alarms/file/{name}` for a single file (2.7.9.055 and later), `PUT /admin/alarms/enabled` to toggle, and `POST /admin/alarms/reload` to re-read from disk. This is the route for version-controlled alarm definitions.
- **The files** — edited directly, then reloaded.

5.6 What a raised alarm looks like

Every change is written to the alarm log as one JSON record:

```
{
  "AlarmName": "EVENT-REPORT-ABC-1200",
  "Description": "REPORT-ABC-1200 lateness",
  "CurrentStatus": "FAIL",
  "Date-Time": "20260928.121001",
```

```
  "Comment": "Synchro Threshold Breached",
  "Component": "REPORT-ABC-1200",
  "Type": "SynchroEvents",
  "Metric": "LATE-S",
  "Target": "Report",
  "NewAlarmValue": "3",
  "OldAlarmValue": 2,
  "DTIME": "1",
  "LAST": 600.0, "MAX": 600.0, "MIN": 0.0,
  "AVERAGE": 18.78, "SUM": 180300.0, "COUNT": 9598
}
```

The record carries the statistics that were current when it fired, so the log is self-contained: it says what happened without requiring a query to work out why. `OldAlarmValue` and `NewAlarmValue` give the transition, which is what distinguishes a new problem from one that is deteriorating.

`CurrentStatus` appears abbreviated in the log — `WARN`, `CRIT`, `FAIL`.

5.7 Alarms are time series

An alarm's severity over time is stored like any other series, which has three useful consequences:

- Alarm history can be charted next to the data that caused it
- Alarms can be aggregated — "how many Critical alarms across this estate?"
- An alarm can watch another alarm, which is how you detect a rule that has been flapping all week

5.8 Where alarms appear

Place	Shows
Alarm status strip	counts by severity, always visible in SynchroUI
Alarm bell	notification of recent changes
Alarms panel	current state within a view
Graph threshold bands	the OK/Warning/Critical zones drawn behind a chart
`GET /alarms/state`	current state, for your own tooling
Alarm log	the full record of every change

5.9 Alarms worth having on every installation

Experience suggests a small set that most installations want:

Silence on each data source. A `DTime` rule per component. The single most valuable alarm, and the one most often missing.

Filesystem usage on the Synchro server itself. Synchro will fill the disk it writes to, and when it does, it stops recording — including the evidence of why. `FS_USAGE-PCT` on the server's own filesystem, at eighty percent.

Memory and commit on monitored hosts. `HOST_MEM_HOST` catches the ordinary case; on Windows, `HOST_COMMITTED-PCT_HOST` catches the case where allocations fail while physical memory still looks free.

Process disappearance. A `Count` or `DTime` rule on `PROC_CPU_<name>` for processes that must always be running.

6. Installation and administration

6.1 SynchroServer

SynchroServer is distributed as a `.tar.gz` containing an installer. Unpack it, run `install.sh`, and start the server with the supplied scripts. It runs as an ordinary user account and needs no root.

Consult the release notes accompanying a given version for its prerequisites, and HTTP and HTTPS Configuration for putting it behind TLS, which is strongly recommended for any server reachable beyond a trusted network.

6.2 Agents

Install SynchroAgent on each monitored host and point it at the server. See the SynchroAgent User Guide.

Each host must report under a distinct **component** name. The agent defaults to the host name, which is usually right, but cloud machines often have generated names and several probes may run on one host, so the name can be set explicitly.

6.3 Users and passwords

User accounts are managed in SynchroUI, or through `/admin/users`. Passwords are hashed, subject to a policy, and login attempts are rate limited. Default credentials shipped with a new installation must be changed before the server is exposed to anything.

Password Management covers storage, policy, migration of existing installations and recovery.

6.4 Retention, compression and aggregation

Three settings govern how much history is kept and at what cost:

Setting	Governs
Retention	how long data is kept before deletion
Compression	when older data is compacted on disk
Aggregation	how series are rolled up over longer spans

They are configured through `/admin/config/{name}` or SynchroUI. Because Synchro keeps second-level data by default, retention is the setting with the largest effect on disk consumption — worth reviewing before a new installation accumulates a year of history rather than after.

> **Monitor the server's own filesystem.** Synchro will happily fill the disk it is writing to. An alarm on the server's own `FS_USAGE-PCT` costs nothing and prevents the one outage that stops all the others being recorded.

6.5 The legacy administration interface

Installations upgraded from older versions may still carry the legacy administration interface. **It is disabled by default from 2.7.9.054 onward** and should stay that way: administration belongs in SynchroUI and the REST API, both of which authenticate properly.

If an existing workflow still depends on it, see Legacy Admin Interface, and treat re-enabling it as a temporary measure.

Appendix A. Metric naming

Names reaching the server are normalised, and it is easier to generate names that survive than to debug ones that do not.

Rule	Consequence
Keys split on _ into exactly three parts	a fourth underscore breaks the routing
Allowed characters: `a–z A–Z 0–9 @ . _ = : ; – ~`	anything else becomes –
Targets are truncated	long device or adapter names appear clipped
Values at or below the emission threshold are not sent	an idle device produces no series at all, which is normal

The last point is worth remembering when a metric appears to be missing: in Synchro, absence usually means nothing happened rather than something failed.

Appendix B. Sizing and performance

A single Synchro infrastructure sustains more than 200,000 updates per second on commodity hardware. Reaching that depends on a few things:

Interval. Every measurement is taken on one cycle. A one-second interval across a large estate is the most common cause of load, on both the agents and the server. Lengthening it is the bluntest lever available.

Series count, not update rate. The cost of a host is dominated by how many distinct series it reports. Per-process metrics typically outnumber everything else on a machine by an order of magnitude, so disabling them is the single largest reduction available where a host does not need that detail.

Retention. Second-level data for years is supported, but it is a choice with a disk cost. Decide it deliberately.

Disk throughput. Synchro is write-heavy by nature. Where an installation struggles, the storage is a more likely cause than the processor.