# SNMPeek Technical Reference

Complete Process Flow & Architecture

Josh Tunnissen

February 2026

## Contents

# 1. Overview

SNMPeek is a Python CLI tool that queries an AKiPS network monitoring server to search for and report on network devices. It auto-detects the type of search the user intends based on the input string and routes the query through the appropriate pipeline.

**Supported query types (auto-detected):**

| Input Example | Detected Type | Pipeline Used |
|---|---|---|
| `10.1.0.0/24` | CIDR Subnet | Host Query (api-db) |
| `192.168.1.1` | Single IP | Host Query (api-db) |
| `switch-core` | Hostname | Host Query (api-db) |
| `*router*` | Hostname wildcard | Host Query (api-db) |
| `aa:bb:cc:dd:ee:ff` | Full MAC address | SPM Query (api-spm) |
| `00:50:56` | Partial MAC address | SPM Query (api-spm) |
| `aabb.ccdd` | Partial MAC (Cisco) | SPM Query (api-spm) |

Multiple queries of mixed types can be passed in a single invocation. Each is classified independently and routed to the correct pipeline.

## 1.1 Technology Stack

- **Language:** Python 3.13
- **AKiPS Client:** `akips` Python library (v0.5.1) — wraps the AKiPS HTTP Web API
- **Terminal UI:** `rich` library — formatted tables, panels, spinners
- **Config:** `python-dotenv` — reads `.env` for server credentials

## 1.2 External Dependencies

The tool depends on a running **AKiPS Network Monitoring** server with the following Web API sections enabled:

| API Section | Purpose | Required? |
|---|---|---|
| `api-db` | Device list, ping/SNMP states, LLDP, groups, events | Yes |
| `api-spm` | Switch Port Mapper (MAC lookups) | Only for MAC queries |
| `api-script` | Site script functions | No (not currently used) |
| `api-msg` | Syslog and trap messages | No (not currently used) |

# 2. Complete Process Flow

## 2.1 High-Level Execution Flow

```
main()
  |
  +-- parse_args()                     Parse CLI flags and positional queries
  |
  +-- classify_query()  [per query]    Auto-detect: MAC / subnet / hostname
  |     |
  |     +-- MAC_PATTERN.match()        Full MAC? (12 hex chars, any format)
  |     +-- is_partial_mac()           Partial MAC? (delimited hex fragment)
  |     +-- ipaddress.ip_network()     Valid CIDR / IP?
  |     +-- fallback: hostname         Wrap bare strings in wildcards
  |
  +-- connect_akips()                  Build AKIPS API client from .env
  |
  +-- [IF mac_addresses]               === MAC Pipeline ===
  |     +-- fetch_mac_data()
  |     |     +-- query_spm()          GET /api-spm?mac=...
  |     |     +-- parse_spm_response() Parse delimited SPM text
  |     |     +-- get_attributes()     Enrich: port status/speed/descr
  |     |     +-- get_events()         Enrich: 7-day port event history
  |     +-- display_mac_results()      Rich table + event history table
  |     +-- write_mac_csv()            CSV export
  |
  +-- [IF networks or hostnames]       === Host Pipeline ===
        +-- fetch_data()
        |     +-- get_devices()        All devices + base SNMP attrs
        |     +-- get_attributes()     PING.icmpState for every device
        |     +-- [optional]           Groups, SNMP state, LLDP neighbors
        +-- filter_and_merge()         Match devices against query filters
        +-- display_results()          Rich table with color-coded status
        +-- write_csv()                CSV export
```

## 2.2 Phase 1 — Argument Parsing

**Function:** `parse_args()`

The argument parser accepts one or more positional `query` strings plus optional flags that control which data columns are displayed.

**Positional arguments:**

- `query` (one or more) — CIDR subnet, MAC address, or hostname pattern

**Optional field flags:**

| Flag | Long Form | Effect |
|---|---|---|
| `-g` | `--groups` | Include AKiPS group memberships |
| `-d` | `--descr` | Include SNMPv2-MIB.sysDescr (OS/model) |
|  | `--location` | Include SNMPv2-MIB.sysLocation |
| `-l` | `--lldp` | Include LLDP neighbor information |
|  | `--snmp` | Include SNMP reachability state |
| `-a` | `--all-fields` | Enable all of the above |

**Output flags:**

| Flag | Long Form | Effect |
|---|---|---|
| `-o` | `--output` | Custom CSV filename |
|  | `--no-csv` | Skip CSV generation entirely |

When `--all-fields` is set, the code flips all individual field booleans to `True` so downstream functions only check the individual flags.

## 2.3 Phase 2 — Query Classification

**Function:** `classify_query(query_str)`
**Returns:** `(type_string, normalized_value)`

This is the core routing logic. Each user-supplied query string is tested against a series of patterns in priority order:

### Step 1: Full MAC Address Check

The compiled regex `MAC_PATTERN` tests for a complete 48-bit MAC address in any of five common notations (case-insensitive):

| Format | Example |
|---|---|
| Colon-separated octets | `aa:bb:cc:dd:ee:ff` |
| Dash-separated octets | `aa-bb-cc-dd-ee-ff` |
| Cisco dot notation (3 groups of 4) | `aabb.ccdd.eeff` |
| Dash groups of 4 | `aabb-ccdd-eeff` |

| Format | Example |
|---|---|
| Plain hex (no delimiters) | `aabbccddeeff` |

If matched, `normalize_mac()` strips all non-hex characters, lowercases, and re-joins as colon-separated pairs: `aa:bb:cc:dd:ee:ff`.

**Return:** `("mac", "aa:bb:cc:dd:ee:ff")`

### Step 2: Partial MAC Address Check

**Function:** `is_partial_mac(s)`

If the full MAC regex did not match, the string is tested for partial MAC patterns. A **delimiter is required** to avoid misidentifying short hex hostnames. The following patterns are recognized:

| Pattern | Example | Description |
|---|---|---|
| `XX:XX` … `XX:XX:XX:XX:XX` | `00:50:56` | 2–5 colon-separated octets |
| `XX-XX` … `XX-XX-XX-XX-XX` | `00-50-56` | 2–5 dash-separated octets |
| `XXXX.X` … `XXXX.XXXX` | `0050.56` | Cisco partial (1 quad + fragment) |
| `XXXX-X` … `XXXX-XXXX` | `0050-56` | Dash quad partial |

If matched, `normalize_mac()` processes the fragment the same way (strip delimiters, lowercase, re-pair with colons). A 3-octet OUI like `00:50:56` stays `00:50:56`.

**Return:** `("mac", "00:50:56")`

### Step 3: CIDR / IP Address Check

Python's `ipaddress.ip_network(query_str, strict=False)` is called. The `strict=False` flag allows host bits to be set (e.g. `10.1.0.5/24` is accepted and normalized to `10.1.0.0/24`). A bare IP like `192.168.1.1` becomes a `/32` network.

**Return:** `("subnet", IPv4Network("10.1.0.0/24"))`

### Step 4: Hostname Fallback

Everything that fails the above checks is treated as a hostname pattern. If the string contains no glob wildcards (`*` or `?`), it is automatically wrapped: `"switch"` becomes `"*switch*"` for substring matching. Explicit wildcards are preserved as-is.

**Return:** `("hostname", "*switch*")`

## 2.4 Phase 3 — API Connection

**Function:** `connect_akips()`

Reads credentials from the `.env` file via `python-dotenv`:

| Variable | Purpose | Default |
|---|---|---|
| `AKIPS_SERVER` | Hostname or IP of AKiPS server | *(required)* |
| `AKIPS_USERNAME` | API username | `api-ro` |
| `AKIPS_PASSWORD` | API password | *(required)* |
| `AKIPS_VERIFY_SSL` | Verify TLS certificates | `true` |
| `AKIPS_TIMEZONE` | Server timezone for timestamps | `America/New_York` |

Returns an initialized `AKIPS` client object that holds an authenticated `requests.Session` for all subsequent API calls.

## 2.5 Phase 4A — MAC Address Pipeline

This pipeline executes when one or more queries were classified as `"mac"`.

### 2.5.1 SPM Query

**Function:** `query_spm(api, mac=None, ip=None)`

Calls the AKiPS Switch Port Mapper endpoint:

```
GET https://<server>/api-spm?username=...&password=...&mac=<normalized_mac>
```

The `api._get(section="api-spm", params={"mac": mac})` method is used directly. This is a separate API section from the main `api-db` database and must be independently enabled on the AKiPS server.

**Returns:** Raw text response from the server, or `None` on error.

### 2.5.2 Response Parsing

**Function:** `parse_spm_response(text)`

The SPM API returns delimited text data. The parser auto-detects the format:

1. **Delimiter detection:** Checks the first line for tab characters, then semicolons, then falls back to comma.
2. **Header detection:** If 2 or more fields in the first line contain keywords like “mac”, “vendor”, “switch”, “interface”, “vlan”, or “ip”, the line is treated as a header row and used to map columns by name.
3. **Data parsing:** Each subsequent line is split by the detected delimiter and mapped to a dictionary.

**Without a header**, the assumed field order is:

| Position | Field | Description |
|---|---|---|
| 0 | `mac` | MAC address |
| 1 | `vendor` | OUI vendor / manufacturer |
| 2 | `switch` | Switch device name in AKiPS |
| 3 | `interface` | Physical port name |
| 4 | `vlan` | VLAN name or ID |
| 5 | `ip` | Associated IP address (may be empty) |

### 2.5.3 Port Enrichment

For each SPM result that includes a switch and interface name, the tool fetches additional port-level attributes from the AKiPS database via `api-db`:

```
mget * <switch> <interface> /IF-MIB.ifOperStatus|IF-MIB.ifAlias|
                             IF-MIB.ifHighSpeed|IF-MIB.ifAdminStatus/
```

| Attribute | Type | Data Extracted |
|---|---|---|
| `IF-MIB.ifOperStatus` | Enum | Operational status (up/down) + last-change timestamp |
| `IF-MIB.ifAdminStatus` | Enum | Administrative status (up/down) |
| `IF-MIB.ifAlias` | Text | Interface description (often room/jack/cable ID) |
| `IF-MIB.ifHighSpeed` | Gauge | Port speed in Mbps |

Enum values are parsed by `parse_enum_state()` which extracts the text value and the `modified` epoch timestamp from the AKiPS enum format: `number,value,created_epoch,modified_epoch,description`.

### 2.5.4 Event History

For each switch/interface pair, recent events are fetched:

```
mget event all time last7d <switch> <interface> *
```

This returns up to 20 events covering the last 7 days. Each event includes:

| Field | Description |
|---|---|
| `epoch` | Unix timestamp |
| `parent` | Switch device name |
| `child` | Interface name |
| `attribute` | Attribute that triggered the event |
| `type` | Event classification (critical, enum, threshold, uptime) |
| `flags` | Event flags |
| `details` | Human-readable event description |

### 2.5.5 MAC Results Display

**Function:** `display_mac_results(mac_data_list)`

For each queried MAC address, the output consists of:

**1. Summary Panel** (magenta border):

```
+-- MAC Address Lookup -------------------------+
| MAC: aa:bb:cc:dd:ee:ff  |  Results: 1        |
+-----------------------------------------------+
```

**2. Results Table** (magenta theme):

| Column | Source | Description |
|---|---|---|
| MAC Address | SPM | Full MAC from SPM response |
| Vendor | SPM | OUI manufacturer name |
| IP Address | SPM | Associated IP or em-dash |
| Switch | SPM | AKiPS device name of the switch |

| Column | Source | Description |
|---|---|---|
| Port | SPM | Physical interface name |
| VLAN | SPM | VLAN name/ID |
| Port Status | Enrichment | Operational status (color-coded) |
| Speed | Enrichment | Link speed (e.g. 1G, 10G) |
| Description | Enrichment | IF-MIB.ifAlias value |

Port status is color-coded:

- Green: `up`
- Red: `down`
- Yellow: `admin down` (administratively disabled)

**3. Event History Table** (green theme, only shown if events exist):

Columns: Time, Switch, Port, Type, Details.

### 2.5.6 MAC CSV Export

**Function:** `write_mac_csv(mac_data_list, filename)`

Writes all MAC results to CSV with columns: Query MAC, MAC Address, Vendor, IP Address, Switch, Port, VLAN, Port Status, Admin Status, Speed (Mbps), Port Description, Port Last Change.

Default filename: `akips_mac_<hex>.csv` where `<hex>` is the query MAC with colons removed.

## 2.6 Phase 4B — Host Pipeline (Subnet / Hostname)

This pipeline executes when one or more queries were classified as `"subnet"` or `"hostname"`.

### 2.6.1 Data Fetching

**Function:** `fetch_data(api, args)`

Issues parallel-ish requests to the AKiPS `api-db` section:

| Step | Command | Data Returned |
|---|---|---|
| 1 (always) | `mget text * sys /<attrs>/` | All devices with sysName, sysDescr, sysLocation, ip4addr |
| 2 (always) | `mget * * * PING.icmpState` | Ping status and timestamps for all devices |
| 3 (if `-g`) | Group membership API | Device-to-group mappings |
| 4 (if `--snmp`) | `mget * * * SNMP.snmpState` | SNMP reachability enum per device |
| 5 (if `-l`) | `mget * * * /LLDP-MIB.lldpRemSysName\|lldpRemPortId/` | LLDP neighbor discovery |

Each fetch displays a `rich` spinner status message while the HTTP request is in flight.

### 2.6.2 Filtering and Merging

**Function:** `filter_and_merge(networks, hostname_patterns, devices, ping_states, extras, args)`

Iterates over every device returned by AKiPS and checks whether it matches **any** of the user's query filters (OR logic across all queries):

**Subnet matching:**

```
ip = ipaddress.ip_address(device_ip)
if ip in network:    # network is an IPv4Network object
    matched = True
```

**Hostname matching:**

```
fnmatch.fnmatch(hostname.lower(), pattern.lower())
# Also checks device_name (AKiPS primary key) as fallback
```

For each matched device, a result row is built:

**Default fields (always included):**

| Field | Source | Logic |
|---|---|---|
| `hostname` | `SNMPv2-MIB.sysName` or device key | Display name |
| `ip` | `ip4addr` attribute | Device IP address |

| Field | Source | Logic |
|---|---|---|
| `status` | `PING.icmpState` enum | "Active" if value=up, "Inactive" if down, else "Unknown" |
| `uptime` | `PING.icmpState` modified timestamp | `now - modified` when status is Active |
| `last_seen` | `PING.icmpState` modified timestamp | `now` if Active, else `modified` time |

**Optional fields (flag-dependent):**

| Field | Flag | Source |
|---|---|---|
| `descr` | `-d` | `SNMPv2-MIB.sysDescr` |
| `location` | `--location` | `SNMPv2-MIB.sysLocation` |
| `groups` | `-g` | AKiPS group membership list |
| `snmp_state` | `--snmp` | `SNMP.snmpState` enum value |
| `lldp` | `-l` | LLDP neighbor names and port IDs |

Results are sorted by IP address (ascending) for consistent output.

### 2.6.3 Host Results Display

**Function:** `display_results(results, networks, hostname_patterns, args)`

**1. Summary Panel** (blue border):

```
+-- AKIPS Host Query -------------------------------------------+
| Subnet: 10.1.0.0/24  |  Found: 45  |  Active: 42  |         |
| Inactive: 3                                                   |
+---------------------------------------------------------------+
```

**2. Results Table** (blue theme):

Status is color-coded with bullet indicators:

- Green: `Active`
- Red: `Inactive`
- Yellow: `Unknown`

Uptime is formatted as human-readable durations: `138d 3h 1m`.

### 2.6.4 Host CSV Export

**Function:** `write_csv(results, filename, args)`

Default filename pattern: `akips_hosts_<label>.csv` where label is built from query parameters (CIDR slashes become underscores, wildcards become X/Q).

# 3. Data Flow Diagrams

## 3.1 Query Classification Flow

```
User Input
    |
    v
+-----------------------+
| MAC_PATTERN.match()?  |---Yes---> ("mac", normalized_full_mac)
+-----------------------+
    | No
    v
+-----------------------+
| is_partial_mac()?     |---Yes---> ("mac", normalized_partial_mac)
+-----------------------+
    | No
    v
+-----------------------+
| ip_network() valid?   |---Yes---> ("subnet", IPv4Network)
+-----------------------+
    | No (ValueError)
    v
+-----------------------+
| Has wildcards (* ?)?  |---Yes---> ("hostname", pattern_as_is)
+-----------------------+
    | No
    v
("hostname", "*<input>*")          <-- auto-wrapped for substring match
```

## 3.2 MAC Lookup Data Flow

```
normalize_mac(input)
    |
    v
+----------------------+     HTTPS GET
| AKiPS api-spm        |<--------------- /api-spm?mac=aa:bb:cc:dd:ee:ff
| Switch Port Mapper   |
+----------------------+
    |
    | Delimited text (MAC, Vendor, Switch, Interface, VLAN, IP)
    v
+-------------------------+
| parse_spm_response()    |  Auto-detect delimiter + header
+-------------------------+
    |
    | List of dicts [{mac, vendor, switch, interface, vlan, ip}, ...]
    v
+------------------------------+     HTTPS GET
| Enrichment: get_attributes() |<--- /api-db?cmds=mget * <switch> <iface> /IF-MIB.../
```

```
+------------------------------+
    |
    | + port_status, admin_status, port_speed, port_descr, port_last_change
    v
+------------------------------+     HTTPS GET
| History: get_events()        |<--- /api-db?cmds=mget event all time last7d ...
+------------------------------+
    |
    | + events[] (up to 20 entries)
    v
+---------------------------+
| display_mac_results()     |  Rich table + event history table
+---------------------------+
    |
    v
+---------------------------+
| write_mac_csv()           |  CSV file output
+---------------------------+
```

### 3.3 Host Lookup Data Flow

```
+--------------------+     +---------------------+
| Subnet queries     |     | Hostname queries    |
| [IPv4Network, ...] |     | ["*switch*", ...]   |
+--------------------+     +---------------------+
         \                       /
          v                     v
+------------------------------------+     HTTPS GET (multiple calls)
| fetch_data(api, args)              |<--- /api-db?cmds=mget text * sys /.../
|   get_devices()                    |<--- /api-db?cmds=mget * * * PING.icmpState
|   [opt] get_group_membership()     |<--- /api-db?cmds=... (groups)
|   [opt] get_attributes(SNMP)       |<--- /api-db?cmds=... (SNMP state)
|   [opt] get_attributes(LLDP)       |<--- /api-db?cmds=... (LLDP)
+------------------------------------+
         |
         | devices{}, ping_states{}, extras{}
         v
+------------------------------------+
| filter_and_merge()                 |
|   For each device:                 |
|     - IP in any subnet?            |
|     - hostname matches pattern?    |
|     - Extract ping state/uptime    |
|     - Attach optional fields       |
+------------------------------------+
         |
         | results[] sorted by IP
         v
```

```
+----------------------------+
| display_results()          |  Summary panel + rich table
+----------------------------+
         |
         v
+----------------------------+
| write_csv()                |  CSV file output
+----------------------------+
```

## 4. AKiPS Enum Format

Many AKiPS attributes (ping state, SNMP state, interface status) are stored as **enum** types with a five-field comma-separated format:

```
<number>,<value>,<created_epoch>,<modified_epoch>,<description>
```

| Field | Type | Example | Description |
|---|---|---|---|
| number | int | `0` | Numeric enum index from the MIB |
| value | string | `up` | Human-readable enum value |
| created | epoch | `1706000000` | Unix timestamp when first recorded |
| modified | epoch | `1738972475` | Unix timestamp of last state change |
| description | string | `ping check` | Child object description |

**`parse_enum_state()`** extracts the `value` and converts `modified` to a Python `datetime`. This is used to derive:

- **Status:** `up` = Active, anything else = Inactive
- **Uptime:** `now - modified` (time since last state change to "up")
- **Last Seen:** `now` if Active, else the `modified` timestamp

## 5. MAC Address Normalization

All MAC addresses — regardless of input format — are normalized to **colon-separated lowercase** before being sent to the API or displayed.

| Input | Normalized |
|---|---|
| `AA:BB:CC:DD:EE:FF` | `aa:bb:cc:dd:ee:ff` |
| `AA-BB-CC-DD-EE-FF` | `aa:bb:cc:dd:ee:ff` |
| `AABB.CCDD.EEFF` | `aa:bb:cc:dd:ee:ff` |
| `AABB-CCDD-EEFF` | `aa:bb:cc:dd:ee:ff` |
| `AABBCCDDEEFF` | `aa:bb:cc:dd:ee:ff` |
| `00:50:56` (partial) | `00:50:56` |
| `0050.56AB` (partial) | `00:50:56:ab` |

The algorithm: strip all non-hex characters, lowercase, then join every 2 characters with a colon.

## 6. Error Handling

| Scenario | Behavior |
|---|---|
| Missing `.env` credentials | Prints error, exits with code 1 |
| AKiPS API HTTP error | Exception raised, caught in `main()`, printed, exit 1 |
| AKiPS returns `ERROR:` text | `AkipsError` raised by library |
| SPM section not enabled | `query_spm()` catches exception, prints warning, returns `None` |
| Port enrichment fails | Silently caught; MAC results still shown without enrichment |
| No results found | Yellow "no hosts found" / "no results found" message |
| Mixed query with MAC failure | MAC error printed but host pipeline still runs |

## 7. Output Files

| Query Type | Default Filename Pattern | Example |
|---|---|---|
| Subnet | `akips_hosts_<cidr>.csv` | `akips_hosts_10_1_0_0_24.csv` |
| Hostname | `akips_hosts_<pattern>.csv` | `akips_hosts_XswitchX.csv` |
| MAC | `akips_mac_<hex>.csv` | `akips_mac_aabbccddeeff.csv` |
| Mixed | Separate files per type | Both files generated |

The `-o` flag overrides the filename for the primary query type. If both MAC and host queries are present, `-o` applies to whichever is the sole type; otherwise auto-generated names are used for both to avoid collisions.