WAC WebSocket SDK Library Usage

Python package: wac-ws-sdk
Module import: wac_ws_sdk

Overview

This document explains how to install, import, and use the commands already implemented in the WAC WebSocket SDK library.

Installation

Install from a wheel:

```
python -m pip install /path/to/wac_ws_sdk-X.Y.Z-py3-none-any.whl
```

Verify installation:

```
python -c "import wac_ws_sdk; print('ok')"
```

Public API (What You Can Import)

- Clients: WacClient (async), WacClientSync (sync wrapper)
- Models: DeviceInfo, LogRecord, BackupNum, VerifyMode
- Errors: WacClientError, DeviceError
- Handlers: handle_reg, handle_sendlog, handle_senduser

Use the client methods to send commands; use the handlers to process device â†' server frames and build standard replies.

Command Reference (Reusable via the Library)

All messages are JSON objects. Requests use the 'cmd' field. Replies echo with 'ret' and include 'result': true|false. On failure, replies include 'reason'.
Timestamp format: 'YYYY-MM-DD HH:MM:SS'.
Credential types (backupnum): 0..9 finger slots; 10 password/pin; 11 card; 12 all fingerprints; 13 all credentials; 50 photo.

A) reg Register device

Direction: device to server
Client method: register(nosenduser: Optional[bool] = None)
Handler: handle_reg(payload, remote) -> dict
Request example:

```
{
  "cmd": "reg",
  "sn": "WAC14089464",
  "cpusn": "CPU123456789",
  "devinfo": { "...": "..." },
  "nosenduser": true
}
```

Success reply:

```
{
  "ret": "reg",
  "result": true,
  "cloudtime": "YYYY-MM-DD HH:MM:SS",
  "nosenduser": true
```

```
}
```

Failure reply:
```
{ "ret": "reg", "result": false, "reason": "<why>" }
```

B) sendlog Push logs

Direction: device to server
Client method: send_logs(records, count: Optional[int]=None, logindex: Optional[int]=None)
Handler: handle_sendlog(payload) -> dict
Request example:

```
{
  "cmd": "sendlog",
  "sn": "WAC14089464",
  "count": 2,
  "logindex": 0,
  "record": [
    {
      "enrollid": 101,
      "name": "Alice",
      "time": "2025-08-25 12:05:00",
      "mode": 0,
      "inout": 0,
      "event": 8,
      "temp": null,
      "verifymode": 0,
      "image": null
    },
    {
      "enrollid": 102,
      "time": "2025-08-25 12:06:10",
      "mode": 0,
      "inout": 1,
      "event": 0
    }
  ]
}
```

Success reply:

```
{
  "ret": "sendlog",
  "result": true,
  "count": 2,
  "logindex": 0,
  "cloudtime": "YYYY-MM-DD HH:MM:SS",
  "access": 1,
  "message": "ok"
}
```

Failure reply:
```
{ "ret": "sendlog", "result": false, "reason": "<why>" }
```

C) senduser Device pushes a user

Direction: device to server
Client method: send_user_from_device(enrollid, name, backupnum, record, admin: int = 0)

Handler: handle_senduser(payload) -> dict
Request example:

```
{
  "cmd": "senduser",
  "enrollid": 101,
  "name": "Ali",
  "backupnum": 10,
  "admin": 0,
  "record": 1234,
  "imagepath": null
}
```

Success reply:

```
{ "ret": "senduser", "result": true, "cloudtime": "YYYY-MM-DD HH:MM:SS"
}
```

Failure reply:

```
{ "ret": "senduser", "result": false, "reason": "<why>" }
```

D) setuserinfo Store/update a user on device

Direction: server device
Client method: set_userinfo(enrollid, name, backupnum, record, admin: int = 0)
Minimal request (required fields):

```
{
  "cmd": "setuserinfo",
  "enrollid": 101,
  "name": "Alice",
  "backupnum": 10,
  "record": "1234"
}
```

Full request (with optional admin):

```
{
  "cmd": "setuserinfo",
  "enrollid": 101,
  "name": "Alice",
  "backupnum": 10,
  "admin": 0,
  "record": "1234"
}
```

Variants (PIN, card, template, photo):

```
// Password/PIN (backupnum 10)
{
"cmd":"setuserinfo","enrollid":101,"name":"Ali","backupnum":10,"reco
rd":1234 }

// Card (backupnum 11)
{
"cmd":"setuserinfo","enrollid":101,"name":"Ali","backupnum":11,"reco
rd":0001234567 }

// Fingerprint template (backupnum 0..9, base64)
{
```

```
"cmd":"setuserinfo","enrollid":101,"name":"Ali","backupnum":0,"recor
d":"Rk1QX1RFTVBMQVRFX0JBU0U2NA==" }

// Photo/face (backupnum 50, base64)
{
"cmd":"setuserinfo","enrollid":101,"name":"Ali","backupnum":50,"reco
rd":"iVBORw0KGgoAAAANSUhEUgAA..." }
```

Expected device confirmation:

```
{
  "ret": "setuserinfo",
  "sn": "WAC14089464",
  "enrollid": 101,
  "backupnum": 10,
  "result": true
}
```

Failure confirmation:

```
{
  "ret": "setuserinfo",
  "sn": "WAC14089464",
  "enrollid": 101,
  "backupnum": 10,
  "result": false,
  "reason": "<why>"
}
```

E) deleteuser to Delete credentials

Direction: server â†’ device
Client method: delete_user(enrollid, backupnum)
Examples:

```
// Delete a specific credential (e.g., PIN)
{ "cmd":"deleteuser", "enrollid":101, "backupnum":10 }

// Wipe everything for the user
{ "cmd":"deleteuser", "enrollid":101, "backupnum":13 }
```

Expected device confirmation:

```
{
  "ret": "deleteuser",
  "sn": "WAC14089464",
  "enrollid": 101,
  "backupnum": 13,
  "result": true
}
```

Failure confirmation:

```
{
  "ret": "deleteuser",
  "sn": "WAC14089464",
  "enrollid": 101,
  "backupnum": 13,
  "result": false,
  "reason": "<why>"
```

```
}
```

BackupNum (Credential Types)

```
0..9  Fingerprint slots
10    Password / PIN
11    Card
12    All fingerprints
13    All credentials (wipe all for user)
50    Photo
```

Appendix VerifyMode (Seen in Logs)

```
0  FP
1  CARD
2  PWD
8  FACE
```

Error Model

Replies may contain result:false and a reason:

```
{ "ret": "<cmd>", "result": false, "reason": "<why>" }
```

Client methods raise DeviceError on device-declared failures, and WacClientError for transport/usage issues.