**OCPP 2.0.1 Server Documentation**

**Overview**

This project implements a server for the Open Charge Point Protocol (OCPP) version 2.0.1. It provides a WebSocket server that charging stations can connect to, along with a REST API for monitoring and managing those connections.

**Table of Contents**

* [Architecture](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#architecture)
* [Core Components](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#core-components)
* [Installation](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#installation)
* [Configuration](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#configuration)
* [API Reference](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#api-reference)
  + [WebSocket API](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#websocket-api)
  + [REST API](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#rest-api)
* [Message Handling](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#message-handling)
* [Code Structure](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#code-structure)
* [Error Handling](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#error-handling)
* [Utilities](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#utilities)
* [Extending the Server](https://claude.ai/chat/4fb5a99e-742f-40b6-a147-82457163a9cd#extending-the-server)

**Architecture**

The server uses a layered architecture:

1. **FastAPI Server Layer**: Handles HTTP and WebSocket connections
2. **OCPP Message Handler Layer**: Processes OCPP messages from charging stations
3. **ChargePoint Implementation Layer**: Implements responses to OCPP commands

This design separates concerns and makes it easier to extend or modify individual components.

**Core Components**

**server.py**

This is the main entry point for the FastAPI application. It:

* Sets up the WebSocket endpoint for OCPP connections
* Provides REST API endpoints for monitoring and managing chargers
* Handles connection acceptance and protocol negotiation
* Maintains a list of active connections
* Includes a background task for monitoring connections

**ocpp\_handler.py**

This handles the OCPP connection logic for each charging station:

* Maintains the connection state
* Routes OCPP messages to appropriate handlers
* Manages message queues
* Converts between OCPP message formats and Python objects
* Keeps track of charger status

**chargePoint.py**

This implements the OCPP 2.0.1 protocol:

* Defines handlers for all OCPP operations
* Creates appropriate response objects for each type of request
* Provides methods for sending commands to the charging station

**utils.py**

Provides utility functions:

* Logging setup
* Data conversion utilities
* Error code definitions
* String transformation utilities (camelCase to snake\_case)

**main.py**

Simple entry point script that starts the server using uvicorn.

**Installation**

**Prerequisites**

* Python 3.7 or higher
* FastAPI
* Uvicorn
* OCPP library

**Installation Steps**

1. Clone the repository
2. Install dependencies:

pip install -r requirements.txt

1. Run the server:

python main.py

Alternatively, you can run the server directly:

uvicorn server:app --host 127.0.0.1 --port 8000 --reload

**Configuration**

The server uses default configuration for development purposes:

* Host: 127.0.0.1
* Port: 8000
* CORS: Allows all origins (should be restricted in production)
* Logging: INFO level, writes to both console and file

For production deployments, consider:

* Restricting CORS settings
* Configuring proper SSL/TLS
* Adjusting log levels
* Setting appropriate timeouts

**API Reference**

**WebSocket API**

**Connection Endpoint**

/ocpp/{charger\_id}

When a charging station connects:

1. The server validates that the client is using the "ocpp2.0.1" subprotocol
2. If an existing connection exists for the same charger\_id, it's closed
3. A new connection handler is created and started
4. The connection remains open until explicitly closed or a timeout occurs

**REST API**

**Get Server Info**

GET /

Returns basic information about the server.

**Response:**

{

"name": "OCPP Server",

"version": "1.0.0",

"description": "A FastAPI server implementing OCPP 2.0.1 protocol"

}

**List Connected Chargers**

GET /chargers

Returns a list of all connected charging stations with their status.

**Response:**

[

{

"id": "CHARGER001",

"status": "Available",

"available": true,

"connected\_since": "2023-01-01T12:00:00.000000",

"last\_seen": "2023-01-01T12:05:00.000000"

}

]

**Get Charger Information**

GET /chargers/{charger\_id}

Returns detailed information about a specific charging station.

**Response:**

{

"id": "CHARGER001",

"status": "Available",

"available": true,

"connected\_since": "2023-01-01T12:00:00.000000",

"last\_seen": "2023-01-01T12:05:00.000000"

}

**Reset Charger**

POST /chargers/{charger\_id}/reset

Sends a reset command to a charging station.

**Parameters:**

* reset\_type (query): Type of reset, either "Hard" or "Soft"

**Response:**

{

"message": "Soft reset initiated for charger CHARGER001"

}

**Send Heartbeat to Charger**

POST /chargers/{charger\_id}/heartbeat

Sends a heartbeat request to a charging station.

**Response:**

{

"message": "Heartbeat sent to charger CHARGER001"

}

**Get Variable from Charger**

POST /chargers/{charger\_id}/getvariable

Requests the value of a specific variable from a charging station.

**Parameters:**

* component\_name (query): Name of the component
* variable\_name (query): Name of the variable

**Response:**

{

"message": "GetVariables request sent to charger CHARGER001"

}

**Set Variable on Charger**

POST /chargers/{charger\_id}/setvariable

Sets the value of a specific variable on a charging station.

**Parameters:**

* component\_name (query): Name of the component
* variable\_name (query): Name of the variable
* value (query): Value to set

**Response:**

{

"message": "SetVariables request sent to charger CHARGER001"

}

**Message Handling**

**OCPP Message Types**

The server handles three types of OCPP messages:

1. **CALL (2)**: Requests from the charging station to the server
2. **CALLRESULT (3)**: Responses to requests
3. **CALLERROR (4)**: Error responses

**CALL Message Flow**

When a CALL message is received:

1. The message is parsed and validated
2. The corresponding handler method is identified (e.g., "BootNotification" -> "on\_boot\_notification")
3. The handler is called with the payload
4. The response is converted to a dictionary
5. A CALLRESULT message is sent back

**Error Handling**

If an error occurs during message processing:

1. The error is logged
2. A CALLERROR message is sent with an appropriate error code
3. For unsupported actions, a "NotImplemented" error is returned

**Code Structure**

**OCPPServerConnection Class**

The core class that handles a single charging station connection. Key methods:

* \_\_init\_\_: Initializes the connection
* start: Starts message processing tasks
* stop: Stops message processing tasks
* send\_message: Sends a message to the charging station
* message\_handler: Processes incoming messages
* handle\_call: Handles CALL messages
* handle\_call\_result: Handles CALLRESULT messages
* handle\_call\_error: Handles CALLERROR messages

**ChargePoint201 Class**

Extends the OCPP library's ChargePoint class to implement all required response handlers:

* on\_boot\_notification: Handles boot notification
* on\_status\_notification: Handles status updates
* on\_heartbeat: Responds to heartbeats
* Other handlers for all OCPP 2.0.1 operations

Additionally provides methods for sending requests to the charging station:

* set\_variables\_req: Sends SetVariables requests
* get\_variables\_req: Sends GetVariables requests
* Many other methods for different request types

**Error Handling**

**Error Codes**

The server uses standard OCPP error codes:

* NotImplemented: The requested action is not implemented
* NotSupported: The requested action is not supported
* InternalError: An internal error occurred
* ProtocolError: A protocol error occurred
* SecurityError: A security error occurred
* FormationViolation: Invalid message format
* PropertyConstraintViolation: A property constraint was violated
* OccurrenceConstraintViolation: An occurrence constraint was violated
* TypeConstraintViolation: A type constraint was violated
* GenericError: A generic error occurred

**Exception Handling**

The server includes comprehensive exception handling:

* WebSocket exceptions are caught and logged
* Invalid JSON is detected and reported
* Invalid message formats trigger appropriate errors
* Timeouts disconnect inactive chargers

**Utilities**

**Logging**

The server uses a structured logging approach:

* Separate log files for different components
* Console and file logging
* Timestamps and log levels
* Detailed error logging with stack traces

**Data Conversion**

Functions for converting between data formats:

* convert\_to\_dict: Converts response objects to dictionaries
* camel\_to\_snake: Converts camelCase to snake\_case
* snake\_to\_camel: Converts snake\_case to camelCase

**Extending the Server**

**Adding New OCPP Operations**

To add support for a new OCPP operation:

1. Add a handler method in the ChargePoint201 class:

@on(Action.new\_action)

def on\_new\_action(self, \*\*kwargs):

return call\_result.NewAction(...)

1. If the operation can be initiated by the server, add a method:

async def new\_action\_req(self, \*\*kwargs):

payload = call.NewAction(\*\*kwargs)

return await self.call(payload)

1. If needed, add a REST API endpoint in server.py:

@app.post("/chargers/{charger\_id}/newaction")

async def new\_action(charger\_id: str):

# Implementation

return {"message": "New action initiated"}

**Customizing Response Behavior**

To customize how the server responds to particular OCPP operations:

1. Modify the handler method in the ChargePoint201 class
2. Update the status tracking in the OCPPServerConnection.handle\_call method if needed

**Adding Authentication**

The current implementation does not include authentication. To add it:

1. Implement a authentication middleware for the REST API
2. Add validation during WebSocket connection acceptance
3. Update the connection handler to check credentials

**Conclusion**

This OCPP 2.0.1 server provides a solid foundation for charging station management. It's designed to be extensible and maintainable, with clear separation of concerns between different components.

For production use, consider adding:

* Authentication and authorization
* Persistent storage for connection history
* More detailed monitoring and analytics
* High availability and load balancing
* Proper SSL/TLS configuration