Home Assistant Research: Log Analysis Plugin Development

This document compiles research on Home Assistant architecture, logging system, and custom integration development for creating a log analysis plugin that uses OpenAI models to suggest fixes for common issues.

1. Home Assistant Logging System Structure

Log File Locations

Home Assistant logs are stored in several locations depending on the installation method:

- **Primary Log File**: Located at `/config/home-assistant.log` in the configuration directory
- **Previous Session Log**: Located at `/config/home-assistant.log.1`
- **Docker Logs**: If running in Docker, logs can be accessed via `docker logs --follow MY_CONTAINER_ID`
- **Supervisor Logs**: For Home Assistant OS or Supervised installations, logs are also available in the supervisor's journald

Log Format and Structure

Home Assistant logs follow a standard format:

```
2023-04-20 12:34:56 ERROR (MainThread) [homeassistant.components.sensor] Error while setting up platform for sensor
```

Each log entry typically contains: - Timestamp - Log level (DEBUG, INFO, WARNING, ERROR, CRITICAL) - Thread information - Component path - Message content

Log Configuration

The logging system can be configured in `configuration.yaml`:

```yaml
logger:
  default: info
  logs:
    homeassistant.components.yamaha: critical
    custom_components.my_integration: debug
```

Log levels from most to least severe are: critical, fatal, error, warning, warn, info, debug, and notset.

Accessing Logs

Logs can be accessed through multiple methods: - **User Interface**: Navigate to Settings > System > Logs - **Command Line**: `tail -f /config/home-assistant.log` - **API**: Using the REST API endpoints - **System Log Integration**: Stores information about logged errors and warnings

2. Creating a Custom Integration for Home Assistant

Integration File Structure

Each integration is stored in a directory named after the integration domain. The basic structure is:

```
custom_components/your_domain_name/
├── __init__.py           # Main component file
├── manifest.json         # Integration manifest
├── config_flow.py        # (Optional) For UI configuration
└── const.py              # (Optional) Constants
```

For more complex integrations:

```
custom_components/your_domain_name/
├── __init__.py
├── manifest.json
├── config_flow.py
├── const.py
├── coordinator.py        # For data updates
├── sensor.py             # Entity platform
└── services.yaml         # Service definitions
```

Manifest File

The `manifest.json` file is required and must contain:

```json
{
  "domain": "your_domain_name",
  "name": "Your Integration",
  "codeowners": ["@your_github_username"],
  "dependencies": [],
  "documentation": "https://github.com/yourusername/your_repo",
  "integration_type": "service",
  "iot_class": "local_polling",
  "issue_tracker": "https://github.com/yourusername/your_repo/issues",
  "requirements": [],
  "version": "0.1.0"
}
```

Key fields: - `domain`: Unique identifier for your integration - `name`: Display name - `codeowners`: GitHub usernames of maintainers - `dependencies`: Other integrations required - `documentation`: URL to documentation - `integration_type`: Type of integration (service, hub, device, etc.) - `iot_class`: How the integration interacts with devices - `requirements`: Python packages needed - `version`: Required for custom integrations

Config Flow

To allow users to configure your integration through the UI, create a `config_flow.py` file and add `"config_flow": true` to your manifest.json. This enables setup through the Integrations page.

Integration Types

Home Assistant integrations are categorized by type: - `device`: Single device integration - `entity`: Basic entity platform - `helper`: Helper entities for automation - `hub`: Integration with multiple devices - `service`: Single service integration - `system`: System integration - `virtual`: Points to another integration

For a log analysis tool, `service` is the most appropriate type.

3. Accessing and Parsing Home Assistant Logs Programmatically

Using the REST API

Home Assistant provides a RESTful API that can be used to access logs:

```
import requests

url = "http://your_home_assistant_ip:8123/api/logbook"
headers = {
    "Authorization": "Bearer YOUR_LONG_LIVED_ACCESS_TOKEN",
    "content-type": "application/json",
}

response = requests.get(url, headers=headers)
log_entries = response.json()
```

Relevant API endpoints: - `/api/logbook/<timestamp>`: Retrieves logbook entries - `/api/history/period/<timestamp>`: Fetches historical state changes

Direct File Access

For a custom integration, you can directly access the log files:

```
import os
from homeassistant.core import HomeAssistant

def get_log_file_path(hass: HomeAssistant) -> str:
    """Get the path to the Home Assistant log file."""
    return os.path.join(hass.config.config_dir, "home-assistant.log")

def parse_logs(log_file_path: str):
    """Parse the Home Assistant log file."""
    with open(log_file_path, "r") as log_file:
        for line in log_file:
            # Process each log line
            pass
```

Using the Logger Component

Your integration can interact with the logger component:

```
import logging
from homeassistant.core import HomeAssistant

_LOGGER = logging.getLogger(__name__)

def setup(hass: HomeAssistant, config):
    """Set up the integration."""
    _LOGGER.debug("Setting up integration")

    # Access logs through the hass object
    log_handler = hass.data.get("logger", {}).get("handlers", {}).get("file", None)
    if log_handler:
        log_file_path = log_handler.baseFilename
        # Process log file
```

Using the System Log Integration

The system_log integration stores information about logged errors and warnings:

```
from homeassistant.components import system_log
from homeassistant.core import HomeAssistant

async def get_system_logs(hass: HomeAssistant):
    """Get system logs from the system_log integration."""
    if not system_log.is_loaded(hass):
```

```
        return []

    return await system_log.async_get_system_logs(hass)
```

4. Best Practices for Developing Home Assistant Integrations

Code Organization

- Follow the Home Assistant Development Checklist
- Use the Integration Quality Scale as a guide
- Implement proper error handling and logging
- Use async/await for all I/O operations
- Use the `DataUpdateCoordinator` for efficient data polling

Testing

- Write unit tests for your integration
- Test with different Home Assistant versions
- Test with different Python versions
- Use the Home Assistant development environment for testing

Documentation

- Provide clear documentation in your README.md
- Include installation instructions
- Document configuration options
- Provide examples of usage
- Include troubleshooting information

Security Considerations

- Protect sensitive information (API keys, tokens)
- Validate user input
- Handle errors gracefully
- Implement rate limiting for API calls
- Follow security best practices for Python code

5. Publishing a Home Assistant Integration on GitHub and HACS

GitHub Repository Structure

For HACS compatibility, your repository must follow this structure:

```
/
├── custom_components/
│   └── your_domain_name/
│       ├── __init__.py
│       ├── manifest.json
│       └── [other files]
├── README.md
├── LICENSE
└── hacs.json
```

HACS Requirements

To make your integration available through HACS:

1. Create a `hacs.json` file in the root of your repository:

```
{
  "name": "Your Integration Name",
  "render_readme": true,
  "domains": ["your_domain_name"]
}
```

2. Ensure your `manifest.json` includes all required fields
3. Add your integration to Home Assistant Brands for proper UI integration
4. Use GitHub releases for versioning (recommended)

Publishing Process

1. Create a GitHub repository with the correct structure
2. Implement your integration following best practices
3. Test thoroughly
4. Create releases with semantic versioning
5. Submit your repository to the HACS default repositories by creating a pull request to the HACS default repository

Making Your Integration User-Friendly

- Implement a config flow for easy setup
- Provide clear error messages
- Include diagnostic information
- Implement system health checks
- Add repair functionality for common issues

Conclusion

Creating a Home Assistant integration for log analysis with OpenAI integration requires understanding the Home Assistant logging system, custom integration development, and best practices for publishing. This research provides the foundation for developing a plugin that can analyze logs, identify issues, and suggest fixes using OpenAI's models.

References

- Home Assistant Developer Documentation
- Home Assistant Integration File Structure
- Home Assistant Integration Manifest
- Home Assistant REST API
- HACS Documentation
- Home Assistant Logger Integration
- Home Assistant System Log Integration