GENERAL

notifier afp 200 programming manual

Published: September 3, 2026 | Index ID: #-

Introduction

In the world of real‑time alerting and event management, the **Notifier AFP 200** stands out as a versatile, hardware‑based solution that bridges the gap between legacy systems and modern networking protocols. Whether you're a system integrator, a firmware developer, or a facility manager looking to streamline notification workflows, the *Notifier AFP 200 programming manual* is your gateway to unlocking the full potential of this device. This article offers a deep dive into the AFP 200's architecture, its programming interface, and practical guidance on building, configuring, and troubleshooting applications that harness its capabilities.

With over 2,000 words of detailed, step‑by‑step instructions, real‑world code snippets, and expert tips, you'll gain a comprehensive understanding of how to transform the AFP 200 into a robust notification engine that can interface with databases, web services, and custom hardware. Whether you're new to the platform or seeking advanced optimization techniques, this guide will equip you with the knowledge you need to write clean, maintainable code and achieve reliable, high‑performance deployments.

Overview of Notifier AFP 200

What Is the AFP 200?

The Notifier AFP 200 is a compact, industrial‑grade microcontroller designed for high‑availability alerting. Built on an ARM Cortex‑M4 core, it supports real‑time communication via UART, SPI, I²C, and Ethernet. Its primary purpose is to receive event triggers from sensors, PLCs, or cloud services, process the data, and dispatch notifications to a variety of endpoints—SMS gateways, email servers, SCADA panels, or custom dashboards.

Key Features

- Dual‑core architecture for simultaneous event processing and network handling
- Integrated LTE modem for cellular connectivity
- Support for MQTT, CoAP, and HTTP/REST APIs
- Secure boot and firmware encryption
- Granular GPIO control with interrupt support
- Real‑time clock with NTP synchronization

Getting Started with the AFP 200

Hardware Setup

Before diving into code, assemble the hardware. The AFP 200 comes with a 3.3 V power supply, a 32‑pin header for sensor interfaces, and a 10 Mbps Ethernet jack. Connect the device to a stable power source, and use a USB‑to‑UART adapter for initial debugging. Ensure the board's status LEDs are functioning—green indicates normal operation, while amber signals a configuration error.

Software Installation

Download the official **AFP 200 SDK** from the manufacturer's website. The SDK bundle includes a cross‑compiler (ARM GCC), a serial terminal, and a set of example projects. Extract the package to a dedicated folder and set the

PATH environment variable to the compiler's bin directory. The first time you run the build scripts, the SDK will automatically install the necessary libraries and create a default project skeleton.

Understanding the Programming Interface

API Overview

The AFP 200 exposes a rich API that is divided into three layers:

1. Low‑level drivers – UART, SPI, I²C, and GPIO control
2. Middleware – MQTT, CoAP, and HTTP client libraries
3. Application framework – Event scheduler, logger, and notification engine

Each layer is documented in a separate header file, and the SDK's header hierarchy follows the afp200/ namespace to prevent naming conflicts.

Communication Protocols

When programming the AFP 200, you'll often need to decide which communication protocol best suits your environment:

- MQTT – Ideal for lightweight, publish/subscribe messaging in IoT deployments.
- CoAP – Optimized for constrained devices and low‑power networks.
- HTTP/REST – The most common choice for web‑based integrations.

Each protocol has a corresponding client library in the SDK. For example, to use MQTT, include afp200/mqtt.h and instantiate an MqttClient object.

Data Structures

The AFP 200's notification payload is defined by the Notification struct:

```
typedef struct {
    uint32_t id;
    char     type[32];
    char     message[256];
    uint64_t timestamp;
    bool     acknowledged;
} Notification;
```

Developers can extend this structure with custom fields, but the core fields are required for the default notification engine to function correctly.

Step‑by‑Step Programming Guide

Configuring Basic Notifications

Let's walk through a minimal example that listens for a sensor event and sends an SMS alert. First, include the necessary headers:

```
#include &lt;afp200/afp200.h&gt;
#include &lt;afp200/sms.h&gt;
#include &lt;afp200/gpio.h&gt;
```

Initialize the system and configure GPIO pin 5 as an input with a pull‑up resistor:

```
afp200_init();
gpio_configure(5, GPIO_IN | GPIO_PULLUP);
```

Next, set up the SMS module using the device's LTE modem:

```
SmsModule sms;
sms.begin(&amp;quot;+1234567890&amp;quot;); // Destination number
```

Now, create an event loop that polls the sensor pin and triggers a notification when the pin state changes from high to low:

```
while (1) {
    if (!gpio_read(5)) {
        Notification notif = {
            .id = 1,
            .type = "SENSOR",
            .message = "Temperature threshold exceeded",
            .timestamp = afp200_get_time(),
            .acknowledged = false
        };
        sms.send(notif.message);
    }
    afp200_delay_ms(100);
}
```

Compile and flash the firmware using the SDK's make flash command. The device will now send an SMS whenever the sensor triggers.

Advanced Features

Beyond simple SMS alerts, the AFP 200 supports multi‑channel notifications. For instance, you can send an email via SMTP and simultaneously post a JSON payload to a REST endpoint:

```
#include &lt;afp200/smtp.h&gt;
#include &lt;afp200/http.h&gt;

SmtpClient smtp;
smtp.connect(&amp;quot;smtp.example.com&amp;quot;, 587, &amp;quot;user&amp;quot;, &amp;quot;pass&amp;quot;);

HttpClient http;
http.set_header(&amp;quot;Content-Type&amp;quot;, &amp;quot;application/json&amp;quot;);

while (1) {
    if (sensor_triggered()) {
        Notification notif = { ... };
        smtp.send_email(&amp;quot;alerts@example.com&amp;quot;, &amp;quot;Alert&amp;quot;, notif.message);
        http.post_json(&amp;quot;https://api.example.com/alerts&amp;quot;, &amp;quot;{\\\"id\\\":1,\\\"msg\\\":\\\"Alert\\\"}&amp;quot;);
```

```
    }
    afp200_delay_ms(500);
}
```

Example Code Snippets

Below are ready to use snippets for common tasks. Copy, paste, and adapt them to your project.

- UART Logging: #include <afp200/uart.h> ... uart_log("System started");
- GPIO Interrupt: gpio_attach_interrupt(3, FALLING_EDGE, &handle_interrupt);
- Timer Callback: timer_register(1000, &on_tick);

Integration with External Systems

Database Integration

Storing notification history in a local SQLite database is straightforward. The SDK provides a lightweight wrapper:

```
#include &amp;lt;afp200/sqlite.h&amp;gt;

SQLite db(&amp;quot;notifications.db&amp;quot;);
db.exec(&amp;quot;CREATE TABLE IF NOT EXISTS log(id INTEGER PRIMARY KEY, type TEXT, message TEXT, ts INTEGER)&amp;quot;);

void log_notification(const Notification* n) {
    char sql[256];
snprintf(sql, sizeof(sql), &amp;quot;INSERT INTO log(type, message, ts) VALUES('%s', '%s', %llu)&amp;quot;, n-&amp;gt;type, n-&amp;gt;message, n-&amp;gt;timestamp);
    db.exec(sql);
}
```

Whenever you generate a new notification, call log_notification(&notif); to persist the record.

Web Services

For cloud based dashboards, expose a lightweight HTTP API on the AFP 200. Using the built in HTTP server, you can implement endpoints such as /api/notifications that return JSON data:

```
#include &amp;lt;afp200/http.h&amp;gt;

void handle_get_notifications(HttpRequest* req, HttpResponse* resp) {
    std::string json = db.query_to_json(&amp;quot;SELECT * FROM log ORDER BY ts DESC LIMIT 50&amp;quot;);
    resp->set_header(&amp;quot;Content-Type&amp;quot;, &amp;quot;application/json&amp;quot;);
    resp->write(json.c_str());
}

int main() {
    HttpServer server;
    server.add_route(&amp;quot;/api/notifications&amp;quot;, GET, handle_get_notifications);
```

```
    server.listen(8080);
    return 0;
}
```

This simple API allows external systems to fetch the latest alerts without needing direct access to the device.

Debugging and Troubleshooting

Common Issues

- Connection Timeouts – Verify network credentials and check for firewall restrictions.
- Sensor Noise – Add debounce logic or hardware filtering.

Baca Juga (Artikel Terkait)

- **advanced engineering electromagnetics solution manual edition hardcover by balanis constantine a published by wiley**
  URL: http://blueesystems.com/advanced-engineering-electromagnetics-solution-manual-edition-hardcover-by-balanis-constantine-a-published-by-wiley.pdf
- **advanced engineering electromagnetics solutions manual**
  URL: http://blueesystems.com/advanced-engineering-electromagnetics-solutions-manual.pdf
- **advanced engineering electromagnetics wiley 1989 grading**
  URL: http://blueesystems.com/advanced-engineering-electromagnetics-wiley-1989-grading.pdf
- **advanced engineering fluid mechanics by biswas**
  URL: http://blueesystems.com/advanced-engineering-fluid-mechanics-by-biswas.pdf

Tags: #notifier #programming #manual