Активная акустическая система Bluesound PULSE CINEMA MINI (P431)

Класс: компактный многоканальный стриминг-процессор для пост-продакшн и профессионального мониторинга

Данная акустическая система разработана компанией Bluesound (подразделение Lenbrook Group, Канада) для профессионального и коммерческого применения. Система не предназначена для использования в жилых помещениях в качестве бытового аудиооборудования.

ЦЕЛЕВОЕ НАЗНАЧЕНИЕ (ПРОФЕССИОНАЛЬНОЕ НЕБЫТОВОЕ ПРИМЕНЕНИЕ)

Акустическая система Bluesound PULSE CINEMA MINI (P431) **предназначена исключительно для профессионального небытового использования** в помещениях с ограниченным пространством.

Основные профессиональные сценарии применения:

Сфера применения	**Тип задач**
Малые студии пост-продакшн	Контроль микса для кинопоказа и стриминговых платформ (Dolby Atmos)
Монтажные комнаты (edit suites)	Предварительное сведение и оценка совместимости
Студии дубляжа и озвучивания	Речевой мониторинг с поддержкой виртуальных высотных каналов
Передвижные телевизионные станции (OB vans)	Мобильный профессиональный контроль

Сфера применения	Тип задач
Демонстрационные зоны (showrooms)	Профессиональная демонстрация контента в форматах высокой четкости
Коммерческие AV-инсталляции	Системы озвучивания в барах, ресторанах, отелях, выставочных залах
Образовательные учреждения	Лаборатории аудиотехники, классы звукорежиссуры

Ключевые отличия от модели PULSE CINEMA (P530):

Параметр	PULSE CINEMA (P530)	PULSE CINEMA MINI (P431)
Конфигурация	3.2.2 (16 динамиков)	2.1 (8 динамиков)
Мощность	500 Вт	280 Вт
Ширина	1200 мм	850 мм
Вес	6,69 кг	5,20 кг
Целевое применение	Широкие инсталляции	Компактные помещения

Категорически не предназначено для: бытового использования в жилых помещениях.

Ключевые особенности для профессионального применения:

1. **Dolby Atmos с виртуализацией высоты:** Поддержка Dolby Atmos, Dolby TrueHD, Dolby Digital Plus, LPCM с программной виртуализацией высотных каналов для контроля пространственного аудиоконтента .
2. **Компактный форм-фактор для профессиональных инсталляций:** Оптимальное решение для монтажных комнат и передвижных студий с ограниченным пространством .
3. **Программируемость через Python API:** Полная поддержка BluOS API для автоматизации и интеграции в профессиональные системы управления.

4. **Hi-Res Audio:** Поддержка MQA, DSD256, FLAC, ALAC, WAV, AIFF с частотой дискретизации до 192 кГц / 24 бит .
5. **Профессиональные системные интеграции:** Поддержка Crestron, Control4, RTI, Nice, URC, Lutron .

---

1. ВВЕДЕНИЕ И ОБЩИЕ СВЕДЕНИЯ

Настоящая инструкция предназначена для **квалифицированного персонала**, использующего активную акустическую систему Bluesound PULSE CINEMA MINI (P431) в профессиональных целях. Система является компактным многоканальным сетевым стримером с возможностью программного управления через API .

Основные технические характеристики:

Параметр	**Значение**
Модель	P431 / P431BLKUNV
Конфигурация	2.1 канала с виртуализацией Dolby Atmos
Количество динамиков	8 (2 твитера + 2 СЧ-динамика + 2 вуфера + 2 пассивных радиатора)
Мощность усилителя	280 Вт (интеллектуальный DSP усилитель)
Динамики (твитер)	0,75″ (21 мм) × 2 шт., мощность 38 Вт каждый
Динамики (СЧ)	1,5″ × 3″ (45 мм) × 2 шт., мощность 38 Вт каждый
Вуферы	4″ (102 мм) × 2 шт., мощность 65 Вт каждый
Пассивные радиаторы	4″ (102 мм) × 2 шт.
THD+N	0,030%
Частота дискретизации	до 192 кГц

Параметр	Значение
Разрядность	16–24 бит
Частотная характеристика	20 Гц – 20 кГц (+0.25, -0.76 дБ)
Процессор	ARM Cortex-A53, Quad-Core, 1.8 ГГц

Подключения :

Тип	Назначение
HDMI eARC	Основное подключение к источнику
Оптический (Toslink)	Цифровой вход
Аналоговый (RCA)	Линейный вход
USB Type-A	Внешние накопители (FAT32/NTFS)
SUBW OUT (RCA)	Активный сабвуфер (90 Гц кроссовер)
LAN (Ethernet)	Gigabit (1000 Мбит/с)
Wi-Fi	802.11ac (Wi-Fi 5), 2.4/5 ГГц
Bluetooth	5.2 aptX Adaptive (двунаправленный)

---

2. ТРЕБОВАНИЯ БЕЗОПАСНОСТИ

Перед эксплуатацией системы ознакомьтесь со следующими требованиями:

- **Электропитание:** Используйте прилагаемые кабели питания (в комплекте предусмотрены кабели для 120В и 230В). Выберите кабель, соответствующий вашему региону .
- **Влажность:** Не допускайте попадания жидкости на корпус. Не эксплуатируйте вблизи воды.
- **Вентиляция:** Обеспечьте свободный доступ воздуха. При установке на стену используйте прилагаемый кронштейн и шаблон для точного монтажа .
- **Температура:** Эксплуатируйте при температуре от 0 до 40°C .
- **Температура хранения:** от -10°C до 50°C, влажность 20-80% .
- **Чистка:** Используйте только сухую мягкую ткань. Не применяйте абразивные средства.

⚠ **ПРОФЕССИОНАЛЬНОЕ ПРИМЕЧАНИЕ:** Система является сетевым аудиоустройством и зависит от правильно настроенной локальной сети. Требуется доступ в Интернет для обновлений ПО и использования облачных сервисов .

---

3. РАСПАКОВКА И ПРОВЕРКА КОМПЛЕКТАЦИИ

1. Извлеките компоненты из упаковки. Сохраните оригинальную упаковку для транспортировки.
2. Проверьте комплектацию :

Компонент	Описание
PULSE CINEMA MINI (P431)	Основной блок
Кабель питания 120 В	Для регионов с напряжением 120 В
Кабель питания 230 В	Для регионов с напряжением 230 В
Кабель Ethernet	Для проводного подключения к сети
Кабель HDMI	Для подключения eARC
Кабель оптический	Toslink кабель
Кронштейн для настенного монтажа	С крепежом

Компонент	Описание
Шаблон для настенного монтажа	Для точной разметки
Болты M6 для крепления	2 шт.
Стопорные язычки (L и R)	Для дополнительной фиксации
Шестигранный ключ	Для монтажа
Руководства	Краткое руководство, гарантия, инструкция по безопасности

3. Осмотрите корпус на предмет повреждений, полученных при транспортировке.

---

4. УСТАНОВКА И РАЗМЕЩЕНИЕ (ПРОФЕССИОНАЛЬНАЯ КОНФИГУРАЦИЯ)

4.1. Габаритные размеры

Вариант установки	Размеры (Д × В × Г)
Настольная (на ножках)	850 × 74 × 140 мм
Настенный монтаж	850 × 140 × 84,5 мм

Вес: 5,20 кг

Максимальное энергопотребление: 65 Вт

4.2. Варианты размещения

Система оснащена **встроенным акселерометром**, который автоматически определяет ориентацию (настенная или настольная) и соответствующим образом переназначает левые и правые аудиоканалы .

A. Настольная установка:

- Установите систему горизонтально на резиновые ножки
- Интерфейс верхней панели должен быть обращен к задней части продукта
- Акселерометр автоматически настроит каналы для настольного размещения

Б. Настенный монтаж:

- Используйте прилагаемый кронштейн и шаблон для разметки
- Крепление возможно только в одной ориентации (с учетом наклонных динамиков)
- Система должна монтироваться максимально близко к стене
- Акселерометр автоматически переназначит каналы для настенного размещения

В. Универсальный монтаж:

- Совместима с универсальными и артикулирующими кронштейнами
- Используйте стопорные язычки (Mount Locking Tabs) для дополнительной безопасности
- Язычки обозначены: 'R' (правый) и 'L' (левый), крепятся с задней стороны кронштейна в Y-образные пазы

4.3. Размещение для Dolby Atmos

- Разместите систему **ниже источника** на уровне ушей оператора
- Ориентируйте место прослушивания прямо напротив экрана
- Избегайте препятствий между системой и точкой прослушивания
- Виртуализатор высоты Dolby Atmos не требует специальной акустической обработки помещения

---

5. ПРОГРАММИРОВАНИЕ НА PYTHON (BLUOS API)

5.1. Общие сведения

Bluesound PULSE CINEMA MINI поддерживает управление через **BluOS API** — RESTful веб-сервис, полностью идентичный API модели P530. API доступен через HTTP-запросы к встроенному веб-серверу устройства.

Базовые параметры подключения:

- **Протокол:** HTTP
- **Порт:** 80
- **IP-адрес:** IP-адрес устройства в локальной сети
- **Формат ответа:** XML или JSON (зависит от метода)

5.2. Определение IP-адреса устройства

Перед началом программирования необходимо определить IP-адрес PULSE CINEMA MINI в сети:

```
python
import socket
import requests
import xml.etree.ElementTree as ET

def discover_bluesound_devices():
    """
    Обнаружение Bluesound устройств в локальной сети
    использует UPnP обнаружение (SSDP)
    """
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM, socket.IPPROTO_UDP)
    sock.settimeout(5)

    MCAST_ADDR = "239.255.255.250"
    MCAST_PORT = 1900

    search_message = (
        "M-SEARCH * HTTP/1.1\r\n"
        "HOST: 239.255.255.250:1900\r\n"
        "MAN: \"ssdp:discover\"\r\n"
        "MX: 3\r\n"
        "ST: urn:schemas-upnp-org:device:MediaRenderer:1\r\n\r\n"
    ).encode()

    sock.sendto(search_message, (MCAST_ADDR, MCAST_PORT))

    devices = []
```

```
    try:
        while True:
            data, addr = sock.recvfrom(1024)
            response = data.decode()
            if "Bluesound" in response or "bluesound" in response.lower():
                for line in response.split('\r\n'):
                    if line.lower().startswith('location:'):
                        location = line.split(':', 1)[1].strip()
                        devices.append({'ip': addr[0], 'location': location, 'model': 'P431'})
                        break
    except socket.timeout:
        pass
    finally:
        sock.close()

    return devices

def get_device_info(ip_address: str) -> dict:
    """
    Получение подробной информации об устройстве
    """
    try:
        response = requests.get(f"http://{ip_address}:80/Status", timeout=5)
        # Парсинг ответа в формате key=value
        info = {}
        for line in response.text.strip().split('\n'):
            if '=' in line:
                key, value = line.split('=', 1)
                info[key] = value
        return info
    except Exception as e:
        return {'error': str(e)}

# Пример использования
devices = discover_bluesound_devices()
for device in devices:
    print(f"Найдено устройство P431: {device['ip']}")
    info = get_device_info(device['ip'])
    print(f"  Информация: {info.get('modelName', 'Unknown')}")
```

5.3. Базовые API запросы

BluOS API использует HTTP GET запросы к эндпоинту /Sync:

python
```python
import requests
import json
from typing import Dict, Any, Optional

class BluesoundController:
    """
    Класс для управления Bluesound PULSE CINEMA MINI через Python API
    """

    def __init__(self, ip_address: str):
        """
        Инициализация контроллера

        Args:
            ip_address: IP-адрес устройства в локальной сети
        """
        self.ip = ip_address
        self.base_url = f"http://{ip_address}:80"
        self._sync_endpoint = "/Sync"

    def _send_command(self, command: str, params: Optional[Dict] = None) -> Dict[str, Any]:
        """
        Отправка команды к устройству

        Args:
            command: Команда API
            params: Параметры команды

        Returns:
            Ответ в виде словаря
        """
        url = f"{self.base_url}{self._sync_endpoint}"
        payload = {'cmd': command}
        if params:
            payload.update(params)

        try:
            response = requests.get(url, params=payload, timeout=5)
            response.raise_for_status()

            # API возвращает данные в формате "key=value" на каждой строке
```

```
            result = {}
            for line in response.text.strip().split('\n'):
                if '=' in line:
                    key, value = line.split('=', 1)
                    result[key] = value
            return result
        except requests.exceptions.RequestException as e:
            print(f"Ошибка связи с устройством {self.ip}: {e}")
            return {'error': str(e)}

    # === Управление воспроизведением ===

    def play(self) -> Dict[str, Any]:
        """Запуск воспроизведения"""
        return self._send_command("play")

    def pause(self) -> Dict[str, Any]:
        """Пауза"""
        return self._send_command("pause")

    def stop(self) -> Dict[str, Any]:
        """Остановка"""
        return self._send_command("stop")

    def next_track(self) -> Dict[str, Any]:
        """Следующий трек"""
        return self._send_command("skip")

    def previous_track(self) -> Dict[str, Any]:
        """Предыдущий трек"""
        return self._send_command("back")

    # === Управление громкостью ===

    def set_volume(self, volume: int) -> Dict[str, Any]:
        """
        Установка уровня громкости

        Args:
            volume: Значение громкости (0-100)
        """
        volume = max(0, min(100, volume))
        return self._send_command("volume", {"level": volume})
```

```
def volume_up(self, step: int = 5) -> Dict[str, Any]:
    """Увеличение громкости"""
    current = self.get_volume()
    if 'volume' in current:
        new_volume = min(100, int(current['volume']) + step)
        return self.set_volume(new_volume)
    return {'error': 'Cannot get current volume'}

def volume_down(self, step: int = 5) -> Dict[str, Any]:
    """Уменьшение громкости"""
    current = self.get_volume()
    if 'volume' in current:
        new_volume = max(0, int(current['volume']) - step)
        return self.set_volume(new_volume)
    return {'error': 'Cannot get current volume'}

def mute(self, state: bool = True) -> Dict[str, Any]:
    """
    Управление отключением звука

    Args:
        state: True - отключить звук, False - включить
    """
    command = "mute" if state else "unmute"
    return self._send_command(command)

# === Получение информации ===

def get_volume(self) -> Dict[str, Any]:
    """Получение текущего уровня громкости"""
    return self._send_command("volume")

def get_state(self) -> Dict[str, Any]:
    """Получение состояния проигрывателя"""
    return self._send_command("state")

def get_status(self) -> Dict[str, Any]:
    """Получение полного статуса устройства"""
    return self._send_command("status")

def get_now_playing(self) -> Dict[str, Any]:
    """Получение информации о текущем треке"""
    return self._send_command("NowPlaying")
```

```
# === Управление входами ===

def select_input(self, input_name: str) -> Dict[str, Any]:
    """
    Выбор входного источника

    Args:
        input_name: Имя источника (hdmi, optical, analog, bluetooth)
    """
    input_map = {
        'hdmi': 'hdmi_arc',
        'optical': 'optical',
        'analog': 'analog_in',
        'bluetooth': 'bluetooth'
    }

    mapped_input = input_map.get(input_name.lower(), input_name)
    return self._send_command("select", {"input": mapped_input})

# === Управление режимами звука ===

def set_listening_mode(self, mode: str) -> Dict[str, Any]:
    """
    Установка режима прослушивания [citation:1][citation:4]

    Args:
        mode: Режим (movie, music, night)
    """
    return self._send_command("listeningMode", {"mode": mode})

def set_subwoofer(self, enabled: bool) -> Dict[str, Any]:
    """
    Включение/выключение сабвуфера [citation:1][citation:4]

    Args:
        enabled: True - включен, False - выключен
    """
    state = "1" if enabled else "0"
    return self._send_command("subwoofer", {"state": state})

def set_surround_upmixer(self, enabled: bool) -> Dict[str, Any]:
    """
    Включение/выключение Surround Upmixer [citation:1][citation:4]
```

```
    Args:
        enabled: True - включен, False - выключен
    """
    state = "1" if enabled else "0"
    return self._send_command("surroundUpmixer", {"state": state})

def set_virtualizer(self, enabled: bool) -> Dict[str, Any]:
    """
    Включение/выключение Virtualizer (Dolby Atmos) [citation:1][citation:4]

    Args:
        enabled: True - включен, False - выключен
    """
    state = "1" if enabled else "0"
    return self._send_command("virtualizer", {"state": state})

def set_volume_leveller(self, enabled: bool) -> Dict[str, Any]:
    """
    Включение/выключение Volume Leveller [citation:1][citation:4]

    Args:
        enabled: True - включен, False - выключен
    """
    state = "1" if enabled else "0"
    return self._send_command("volumeLeveller", {"state": state})

# === Управление предустановками ===

def press_preset(self, preset_number: int) -> Dict[str, Any]:
    """
    Активация предустановки [citation:1][citation:4][citation:8]

    Args:
        preset_number: Номер предустановки (1 или 2)
    """
    if preset_number not in [1, 2]:
        return {'error': 'Preset number must be 1 or 2'}
    return self._send_command(f"preset{preset_number}")

def set_preset(self, preset_number: int, value: str) -> Dict[str, Any]:
    """
    Установка предустановки

    Args:
```

```
        preset_number: Номер предустановки (1 или 2)
        value: URL или идентификатор контента
    """
    if preset_number not in [1, 2]:
        return {'error': 'Preset number must be 1 or 2'}
    return self._send_command(f"set{preset_number}", {"value": value})

# === Управление группировкой (multi-room) ===

def group_devices(self, slave_ips: list) -> Dict[str, Any]:
    """
    Группировка нескольких устройств Bluesound

    Args:
        slave_ips: Список IP-адресов устройств для группировки
    """
    result = {}
    for slave_ip in slave_ips:
        group_url = f"http://{slave_ip}:80/Sync?cmd=group&master={self.ip}"
        try:
            response = requests.get(group_url, timeout=5)
            result[slave_ip] = response.status_code == 200
        except requests.exceptions.RequestException as e:
            result[slave_ip] = False
            result[f"{slave_ip}_error"] = str(e)
    return result

def ungroup_devices(self) -> Dict[str, Any]:
    """Отмена группировки"""
    return self._send_command("ungroup")
```

5.4. Примеры использования

Пример 1: Базовое управление P431

python

```
# Создание экземпляра контроллера для PULSE CINEMA MINI
controller = BluesoundController("192.168.1.100")

# Установка громкости для работы в монтажной комнате
controller.set_volume(65)
```

```
# Выбор HDMI входа (подключение к источнику видеоконтента)
controller.select_input("hdmi")

# Установка режима Movie для контроля киноконтента
controller.set_listening_mode("movie")

# Включение Virtualizer для Dolby Atmos
controller.set_virtualizer(True)

# Включение Surround Upmixer для расширения звуковой сцены
controller.set_surround_upmixer(True)

# Воспроизведение
controller.play()

# Получение информации о состоянии
info = controller.get_now_playing()
print(f"Сейчас играет: {info.get('title', 'Неизвестно')}")
print(f"Громкость: {controller.get_volume().get('volume', '?')}%")

# Пауза
controller.pause()
```

Пример 2: Профессиональный мониторинг с логированием

```
python
import time
from datetime import datetime
import csv

class BluesoundProfessionalMonitor:
    """Класс для профессионального мониторинга состояния P431"""

    def __init__(self, ip_address: str, log_file: str = "p431_monitor.csv"):
        self.controller = BluesoundController(ip_address)
        self.log_file = log_file

    def get_complete_state(self) -> dict:
        """Получение полного состояния устройства"""
        state = {
            'timestamp': datetime.now().isoformat(),
            'volume': self.controller.get_volume().get('volume', 'unknown'),
            'state': self.controller.get_state().get('state', 'unknown'),
            'input': self.controller.get_status().get('input_name', 'unknown')
```

```
    }

    # Получение информации о текущем треке
    now_playing = self.controller.get_now_playing()
    if 'title' in now_playing:
        state['title'] = now_playing.get('title', '')
        state['artist'] = now_playing.get('artist', '')
        state('album', '')

    return state

def log_to_csv(self, state: dict):
    """Логирование состояния в CSV файл"""
    file_exists = False
    try:
        with open(self.log_file, 'r'):
            file_exists = True
    except FileNotFoundError:
        pass

    with open(self.log_file, 'a', newline='') as f:
        writer = csv.DictWriter(f, fieldnames=state.keys())
        if not file_exists:
            writer.writeheader()
        writer.writerow(state)

def monitor_session(self, duration_minutes: int = 60, interval_seconds: int = 10):
    """
    Мониторинг профессиональной сессии

    Args:
        duration_minutes: Длительность мониторинга в минутах
        interval_seconds: Интервал между замерами в секундах
    """
    start_time = datetime.now()
    end_time = start_time.timedelta(minutes=duration_minutes)

    print(f"Начало мониторинга сессии в {start_time}")
    print(f"Длительность: {duration_minutes} минут, интервал: {interval_seconds} сек")

    try:
        while datetime.now() < end_time:
            state = self.get_complete_state()
            self.log_to_csv(state)
```

```
            print(f"[{state['timestamp']}] Громкость: {state['volume']}%, Состояние: {state['state']}")
            time.sleep(interval_seconds)
        except KeyboardInterrupt:
            print("\nМониторинг прерван оператором")
        finally:
            print(f"Лог сохранен в {self.log_file}")

    def check_calibration(self, reference_spl_db: int = 85) -> dict:
        """
        Проверка калибровки уровня (требуется SPL метр)

        Args:
            reference_spl_db: Опорный уровень звукового давления
        """
        print(f"=== ПРОЦЕДУРА КАЛИБРОВКИ ===")
        print(f"1. Установите измерительный микрофон в точке прослушивания")
        print(f"2. Воспроизведите калибровочный сигнал (1 кГц, -20 dBFS)")
        print(f"3. Настройте громкость P431 до достижения {reference_spl_db} дБ SPL")
        print(f"4. Зафиксируйте значение громкости")

        current_volume = self.controller.get_volume().get('volume', 'unknown')
        print(f"\nТекущая громкость: {current_volume}%")
        print(f"Рекомендуемая громкость для калибровки: 75-85%")

        return {
            'calibrated': False,
            'reference_spl': reference_spl_db,
            'current_volume': current_volume,
            'recommended_volume': '75-85'
        }

# Пример использования
monitor = BluesoundProfessionalMonitor("192.168.1.100")
calibration = monitor.check_calibration(85)
# monitor.monitor_session(duration_minutes=30, interval_seconds=5)
```

Пример 3: Сценарии автоматизации для пост-продакшн

```
python
class PostProductionAutomation:
    """Автоматизация для пост-продакшн задач"""

    def __init__(self, ip_address: str):
        self.controller = BluesoundController(ip_address)
```

```
def cinema_mix_session(self):
    """Настройка для сессии сведения киноконтента"""
    print("=== НАСТРОЙКА СЕССИИ КИНОМИКСА ===")

    # Выбор HDMI входа (источник с готовым миксом)
    self.controller.select_input("hdmi")

    # Установка Movie режима для киноконтента
    self.controller.set_listening_mode("movie")

    # Включение Virtualizer для Dolby Atmos
    self.controller.set_virtualizer(True)

    # Включение Surround Upmixer для полной сцены
    self.controller.set_surround_upmixer(True)

    # Установка опорной громкости для контроля
    self.controller.set_volume(78)

    print("Конфигурация для киномикса применена")

def dialogue_editing_session(self):
    """Настройка для сессии редактирования диалогов"""
    print("=== НАСТРОЙКА СЕССИИ РЕДАКТИРОВАНИЯ ДИАЛОГОВ ===")

    # Выбор оптического входа (чистый сигнал)
    self.controller.select_input("optical")

    # Music режим для нейтрального воспроизведения диалогов
    self.controller.set_listening_mode("music")

    # Отключение виртуализации для чистого сигнала
    self.controller.set_virtualizer(False)
    self.controller.set_surround_upmixer(False)

    # Более низкая громкость для длительной работы
    self.controller.set_volume(65)

    print("Конфигурация для редактирования диалогов применена")

def quality_control_session(self):
    """Настройка для сессии контроля качества (QC)"""
    print("=== НАСТРОЙКА СЕССИИ КОНТРОЛЯ КАЧЕСТВА ===")
```

```
        # Выбор HDMI входа
        self.controller.select_input("hdmi")

        # Movie режим для оценки
        self.controller.set_listening_mode("movie")

        # Volume Leveller для постоянного уровня
        self.controller.set_volume_leveller(True)

        # Умеренная громкость для объективной оценки
        self.controller.set_volume(72)

        print("Конфигурация для контроля качества применена")

    def reference_listening_session(self, reference_tracks: list):
        """
        Сессия референсного прослушивания

        Args:
            reference_tracks: Список URL референсных треков
        """
        print("=== РЕФЕРЕНСНОЕ ПРОСЛУШИВАНИЕ ===")

        for track in reference_tracks:
            print(f"Воспроизведение: {track}")
            # Здесь может быть логика загрузки и воспроизведения треков
            time.sleep(5)  # Имитация прослушивания

# Пример использования
automation = PostProductionAutomation("192.168.1.100")
automation.cinema_mix_session()
# После сведения для телевидения:
automation.dialogue_editing_session()
```

Пример 4: Асинхронный WebSocket контроллер для реального времени

python

```
import asyncio
import websockets
import json

class BluesoundWebSocketMonitor:
    """
```

```
WebSocket монитор для реального времени
Используется для интеграции с профессиональными системами управления
"""

def __init__(self, ip_address: str):
    self.ip = ip_address
    self.websocket_url = f"ws://{ip_address}:80/ws"
    self.websocket = None
    self.callbacks = []

async def connect(self):
    """Установка WebSocket соединения"""
    try:
        self.websocket = await websockets.connect(self.websocket_url)
        print(f"WebSocket подключен к {self.ip}")
    except Exception as e:
        print(f"Ошибка подключения WebSocket: {e}")

async def subscribe_events(self):
    """Подписка на события устройства"""
    if self.websocket:
        subscribe_msg = json.dumps({"type": "subscribe", "events": ["state", "volume", "track", "input"]})
        await self.websocket.send(subscribe_msg)

async def listen_events(self):
    """Прослушивание событий"""
    if not self.websocket:
        return

    try:
        async for message in self.websocket:
            event = json.loads(message)
            for callback in self.callbacks:
                await callback(event)
    except Exception as e:
        print(f"WebSocket ошибка: {e}")

def add_event_handler(self, callback):
    """Добавление обработчика событий"""
    self.callbacks.append(callback)

async def start_monitoring(self):
    """Запуск мониторинга"""
    await self.connect()
```

```
        if self.websocket:
            await self.subscribe_events()
            await self.listen_events()

# Пример асинхронного использования
async def main_ws():
    monitor = BluesoundWebSocketMonitor("192.168.1.100")

    def handle_event(event):
        print(f"Событие: {json.dumps(event, indent=2)}")

    monitor.add_event_handler(handle_event)
    await monitor.start_monitoring()

# asyncio.run(main_ws())
```

Пример 5: Интеграция с Crestron/Control4 через REST API

```
python
from flask import Flask, request, jsonify
from functools import wraps

app = Flask(__name__)

# Конфигурация устройств P431
P431_DEVICES = {
    "edit_suite_a": "192.168.1.100",
    "edit_suite_b": "192.168.1.101",
    "qc_room": "192.168.1.102",
    "dubbing_studio": "192.168.1.103"
}

# Хранение сессий контроллеров
controllers = {}

def get_controller(device_name: str):
    """Получение или создание контроллера для устройства"""
    if device_name not in P431_DEVICES:
        return None
    if device_name not in controllers:
        controllers[device_name] = BluesoundController(P431_DEVICES[device_name])
    return controllers[device_name]

@app.route('/api/p431/<device_name>/status', methods=['GET'])
```

```python
def get_device_status(device_name):
    """Получение полного статуса устройства"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404

    status = {
        'device': device_name,
        'ip': P431_DEVICES[device_name],
        'model': 'P431',
        'volume': controller.get_volume(),
        'state': controller.get_state(),
        'now_playing': controller.get_now_playing()
    }
    return jsonify(status)

@app.route('/api/p431/<device_name>/volume', methods=['GET', 'POST'])
def handle_volume(device_name):
    """Управление громкостью"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404

    if request.method == 'GET':
        volume = controller.get_volume()
        return jsonify(volume)
    elif request.method == 'POST':
        data = request.get_json()
        if 'level' not in data:
            return jsonify({"error": "Missing 'level' parameter"}), 400
        result = controller.set_volume(data['level'])
        return jsonify(result)

@app.route('/api/p431/<device_name>/play', methods=['POST'])
def handle_play(device_name):
    """Запуск воспроизведения"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404
    result = controller.play()
    return jsonify(result)

@app.route('/api/p431/<device_name>/pause', methods=['POST'])
def handle_pause(device_name):
```

```
    """Пауза"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404
    result = controller.pause()
    return jsonify(result)

@app.route('/api/p431/<device_name>/input', methods=['POST'])
def handle_input(device_name):
    """Выбор входного источника"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404

    data = request.get_json()
    if 'source' not in data:
        return jsonify({"error": "Missing 'source' parameter"}), 400

    result = controller.select_input(data['source'])
    return jsonify(result)

@app.route('/api/p431/<device_name>/listening_mode', methods=['POST'])
def handle_listening_mode(device_name):
    """Установка режима прослушивания"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404

    data = request.get_json()
    if 'mode' not in data:
        return jsonify({"error": "Missing 'mode' parameter"}), 400

    result = controller.set_listening_mode(data['mode'])
    return jsonify(result)

@app.route('/api/p431/devices', methods=['GET'])
def list_devices():
    """Список всех устройств P431"""
    devices_info = {}
    for name, ip in P431_DEVICES.items():
        devices_info[name] = {'ip': ip, 'model': 'P431', 'status': 'configured'}
    return jsonify(devices_info)

@app.route('/api/p431/<device_name>/group', methods=['POST'])
```

```python
def handle_group(device_name):
    """Группировка устройств"""
    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404

    data = request.get_json()
    if 'slaves' not in data:
        return jsonify({"error": "Missing 'slaves' parameter"}), 400

    # Преобразование имен устройств в IP-адреса
    slave_ips = []
    for slave_name in data['slaves']:
        if slave_name in P431_DEVICES:
            slave_ips.append(P431_DEVICES[slave_name])

    result = controller.group_devices(slave_ips)
    return jsonify(result)

@app.route('/api/p431/<device_name>/preset/<int:preset_num>', methods=['POST'])
def handle_preset(device_name, preset_num):
    """Активация предустановки"""
    if preset_num not in [1, 2]:
        return jsonify({"error": "Preset number must be 1 or 2"}), 400

    controller = get_controller(device_name)
    if not controller:
        return jsonify({"error": "Device not found"}), 404

    result = controller.press_preset(preset_num)
    return jsonify(result)

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5001)
```

5.5. Справочник команд BluOS API для P431

Команда	Описание	Параметры
volume	Управление громкостью	level (0-100)

Команда	Описание	Параметры
mute / unmute	Отключение/включение звука	-
play / pause / stop	Управление воспроизведением	-
skip / back	Следующий/предыдущий трек	-
state	Состояние проигрывателя	-
status	Полный статус	-
NowPlaying	Информация о текущем треке	-
select	Выбор входа	input (hdmi_arc, optical, analog_in, bluetooth)
listeningMode	Режим прослушивания	mode (movie, music, night)
subwoofer	Сабвуфер	state (0/1)
surroundUpmixer	Surround Upmixer	state (0/1)
virtualizer	Dolby Atmos виртуализатор	state (0/1)
volumeLeveller	Выравнивание громкости	state (0/1)
preset1 / preset2	Предустановки	-
group	Группировка устройств	master (IP мастер-устройства)
ungroup	Отмена группировки	-

6. ОРГАНЫ УПРАВЛЕНИЯ (СЕНСОРНАЯ ПАНЕЛЬ)

Система оснащена сенсорной панелью с LED-индикацией :

Элемент	Функция
Preset 1 / Preset 2	Две программируемые кнопки предустановок (настраиваются через BluOS App)
Volume + / -	Сенсорные кнопки регулировки громкости
Play/Pause	Многофункциональная кнопка (индикатор сети, воспроизведение/пауза, Mute при группировке)
Status LED	Светодиодный индикатор состояния

Цветовая индикация Status LED :

Код (цвет/режим)	Описание состояния
Короткая вспышка синий → красный	Включение, перезагрузка устройства
Зеленый (постоянный)	Режим «Hotspot» (готовность к подключению)
Зеленый (мигающий)	Подключение к сети
Белая вспышка	Доступно обновление ПО
Красный (постоянный)	Режим обновления
Чередование красный/зеленый	Обновление ПО в процессе
Синий (мигающий)	Режим Mute (только при группировке)

Код (цвет/режим)	Описание состояния
Белый (постоянный)	Индексация (чтение медиатеки)
Синий (постоянный)	Подключен к сети — готов к работе с BluOS App
Красный (мигающий)	Сброс к заводским настройкам (в процессе)
Фиолетовый (постоянный)	Тайм-аут режима Hotspot

---

7. ПОДКЛЮЧЕНИЯ И ИНТЕРФЕЙСЫ

7.1. HDMI eARC

- **Назначение:** Подключение к источнику с поддержкой eARC/ARC
- **Настройка:** Включите ARC/eARC в настройках источника
- **Отображение в приложении:** HDMI ARC или eARC

7.2. Оптический вход (Toslink)

- **Назначение:** Подключение цифровых источников
- **Кабель:** Оптический (в комплекте)
- **Отображение в приложении:** Optical Input

7.3. Аналоговый вход (RCA)

- **Назначение:** Подключение линейного выхода
- **Кабель:** RCA стерео (не входит в комплект)
- **Отображение в приложении:** Analog Input

7.4. Выход сабвуфера (SUBW OUT)

- **Тип разъема:** RCA моно
- **Частота кроссовера:** 90 Гц
- **Альтернатива:** Беспроводное подключение PULSE SUB+

7.5. USB (Type-A)

- **Назначение:** Подключение внешних накопителей
- **Форматирование:** FAT32 или NTFS
- **Режим:** Серверный режим (Server Mode)

7.6. LAN порт (Ethernet)

- **Скорость:** Gigabit (1000 Мбит/с)
- **Кабель:** Ethernet (в комплекте)

---

8. НАСТРОЙКИ ЗВУКА (ЧЕРЕЗ BLUOS APP)

8.1. Режимы прослушивания (Listening Modes)

Режим	Применение
Movie	Оптимизирован для киноконтента, подчеркивает эффекты и диалоги
Music	Нейтральный режим для музыкального контента
Late Night	Снижает динамический диапазон для контроля на низкой громкости

8.2. Функции обработки звука

Функция	Описание
Subwoofer	Включение/выключение выхода сабвуфера (проводного или беспроводного). Частота кроссовера: 90 Гц
Surround Upmixer	Преобразование стерео в виртуальную среду объемного звучания
Virtualizer	Расширение звуковой сцены для любого источника. Включается вместе с Surround Upmixer для максимальной ширины стереопанорамы
Volume Leveller	Поддержание постоянного уровня громкости вне зависимости от выбора источника

8.3. Replay Gain

Опция	Описание
Track Gain	Использует значение громкости трека из метаданных
Album Gain	Использует значение громкости альбома для согласованности
Smart Gain	BluOS автоматически выбирает оптимальную опцию

8.4. Ограничение громкости (Volume Limits)

Установка верхнего и нижнего предела регулировки громкости в децибелах.

---

9. ПРОФЕССИОНАЛЬНЫЕ ФУНКЦИИ

9.1. Автоматическая ориентация (акселерометр)

Встроенный акселерометр автоматически определяет способ установки (настенный или настольный) и переназначает левые и правые аудиоканалы для корректного воспроизведения. **Ручная настройка не требуется.**

9.2. ИК-обучение (IR Learning)

Позволяет обучить систему работать со сторонним ИК-пультом управления.

9.3. Автоматическое включение/выключение

При использовании HDMI eARC система автоматически включается и выключается вместе с источником.

9.4. Беспроводное расширение (Bonded Speaker Link)

Дополнительный Wi-Fi модуль для связи с беспроводными тыловыми динамиками (PULSE M, PULSE FLEX) и сабвуфером (PULSE SUB+) .

9.5. Профессиональные системные интеграции

Поддержка систем управления: **Crestron, Control4, RTI, Nice, URC, Lutron**.

---

10. BLUOS APP (УПРАВЛЕНИЕ)

- **Поддерживаемые платформы:** iOS, Android, Windows, macOS
- **Управление многокомнатной системой (multi-room)**
- **Доступ к 23+ музыкальным сервисам**

- **Воспроизведение локальной медиатеки** (до 200 000 файлов через SMB)

Поддерживаемые аудиоформаты :

Формат	**Поддержка**
MQA	Полная поддержка
DSD	DSD256
FLAC	До 192 кГц / 24 бит
ALAC	До 192 кГц / 24 бит
WAV	До 192 кГц / 24 бит
AIFF	До 192 кГц / 24 бит

Сторонние интеграции :

- AirPlay 2
- Spotify Connect
- TIDAL Connect
- Roon Ready
- Amazon Alexa Skills

---

11. ПОИСК И УСТРАНЕНИЕ НЕИСПРАВНОСТЕЙ

Проблема	Решение
Нет звука	Проверьте питание. Убедитесь, что выбран правильный вход. Проверьте настройки ARC/eARC на источнике
Не удается подключиться к сети	Дождитесь зеленого LED (режим Hotspot). Для Wi-Fi: проверьте пароль. Для Ethernet: проверьте кабель
Нет эффекта Dolby Atmos	Убедитесь, что контент имеет поддержку Dolby Atmos. Включите Virtualizer в настройках
API не отвечает	Проверьте IP-адрес устройства. Убедитесь, что устройство находится в той же подсети
Ошибка Python запроса	Проверьте сетевое соединение. Убедитесь, что порт 80 открыт
Сабвуфер не работает	Проверьте кабель SUBW OUT. Убедитесь, что Subwoofer включен в настройках (Enabled)

---

12. СБРОС К ЗАВОДСКИМ НАСТРОЙКАМ

Процедура (Factory Reset) :

1. Убедитесь, что система включена.
2. Нажмите и удерживайте кнопку **Play/Pause** на верхней панели.
3. Удерживайте до тех пор, пока LED не начнет **мигать красным**.
4. Отпустите кнопку.
5. Система перезагрузится.
    - При беспроводном подключении: LED станет **зеленым** (режим Hotspot)
    - При проводном подключении: LED станет **синим**

Внимание: Сброс удаляет все пользовательские настройки (сеть, предустановки, настройки звука).

13. ТЕХНИЧЕСКОЕ ОБСЛУЖИВАНИЕ

- **Уход за корпусом:** Протирайте мягкой сухой тканью. Алюминиевый корпус и тканевое покрытие требуют бережного обращения .
- **Обновление ПО:** Производится автоматически через BluOS App при подключении к Интернету. Белая вспышка LED указывает на доступность обновления .
- **Транспортировка:** Используйте оригинальную упаковку.
- **Сервис:** При повреждениях обращайтесь к авторизованному дилеру Bluesound.

Гарантийный срок: 24 месяца.

ПРИЛОЖЕНИЕ: ПОЛНЫЕ ТЕХНИЧЕСКИЕ СПЕЦИФИКАЦИИ

Параметр	**Значение**
Модель	PULSE CINEMA MINI (P431)
Артикул	P431BLKUNV
UPC/EAN	786357003245
Цвет	Черный (Black)
Конфигурация динамиков	2.1 канала, 8 динамиков
Габариты (настольная)	850 × 74 × 140 мм
Габариты (настенный)	850 × 140 × 84,5 мм

Параметр	Значение
Вес	5,20 кг
Мощность	280 Вт (38 Вт × 2 ВЧ + 38 Вт × 2 СЧ + 65 Вт × 2 НЧ)
Процессор	ARM Cortex-A53, Quad-Core, 1.8 ГГц
THD+N	0,030%
Частота кроссовера сабвуфера	90 Гц
Частотная характеристика	20 Гц – 20 кГц (+0.25, -0.76 дБ)
API протокол	HTTP REST (порт 80)
Программирование	Python через BluOS API
Макс. энергопотребление	65 Вт
Рабочая температура	0–40 °C
Температура хранения	-10–50 °C