Arinst R3E — **Руководство по протоколу** Alt API

Справочник для разработчиков, желающих взаимодействовать с устройством программно: получать данные спектра, читать информацию об устройстве или настраивать автономные задачи сканирования.

1. **Транспорты**

Один и тот же бинарный протокол (Alt API) доступен одновременно через три транспорта:

Транспорт	Адрес / Порт	Примечания
USB CDC	Виртуальный COM-порт, 115 200 бод	
WebSocket	ws://<ip-устройства>:8080	Бинарные фреймы, один запрос → один ответ
HTTP/HTTPS	http://<ip-устройства>/ порт 80, https:// порт 443	POST-запросы

> **Последовательная модель** — каждое соединение обрабатывает один запрос за раз. Конвейеризация не поддерживается.

IP-адрес устройства по умолчанию: *192.168.1.10* (статический). DHCP и статический IP настраиваются через *API_SET_LWIP_CONF*.

2. **Структура пакета**

Каждый запрос к устройству и каждый ответ используют одинаковый формат: 6-байтовый заголовок + полезная нагрузка + 2-байтовая контрольная сумма.

Смещение	Размер	Поле
0	1	Стартовый байт = 0xBB
1	1	ID команды (см. Раздел 4)
2	1	Длина payload [7:0] (uint16 little-endian)
3	1	Длина payload [15:8]
4	N	Payload (N = длина payload)
4+N	1	CKA (младший байт контрольной суммы)
5+N	1	CKB (старший байт контрольной суммы)

Полный размер фрейма = 6 + N байт (*ALT_API_HEAD_LEN = 6*)

2.1 Алгоритм контрольной суммы (стиль UBX)

Контрольная сумма охватывает все байты со смещения 0 по смещение 3+N (заголовок + payload, без самих байт контрольной суммы).

CK_A = 0
CK_B = 0
для каждого байта B в frame[0 .. 3+N]:

CK_A = (CK_A + B) & 0xFF
CK_B = (CK_B + CK_A) & 0xFF

Фрейм заканчивается байтами *[CK_A, CK_B]* (uint16 little-endian при чтении как слово).

2.2 Особый случай: нулевая контрольная сумма

Если оба байта контрольной суммы в принятом фрейме равны *0x00 0x00*, прошивка пропускает проверку контрольной суммы и обрабатывает пакет безусловно. Удобно для быстрого ручного тестирования.

2.3 Несовпадение контрольной суммы

Если принятая контрольная сумма не совпадает с вычисленной, прошивка принудительно устанавливает *command = API_ERR (0xFC)* и возвращает 6-байтовый ответ об ошибке (без payload).

2.4 Ответ об ошибке

BB FC 00 00 CKA CKB

CKA/CKB вычисляются по 4 байтам заголовка *BB FC 00 00*.

3. Общие типы данных

3.1 uint64 Частота (Гц)

Частоты передаются как 64-**битные беззнаковые целые** little-endian в Гц.

Байты: [B0 B1 B2 B3 B4 B5 B6 B7]
Значение = B0 | (B1<<8) | (B2<<16) | (B3<<24) | ((uint64)B4<<32) | ... | ((uint64) B7<<56)

Пример — 100 МГц = 100 000 000 Гц = *0x05F5E100*:

00 E1 F5 05 00 00 00 00

3.2 float32

Формат IEEE-754 одинарной точности, little-endian, 4 байта.

3.3 **Индекс полосы пропускания** (bw)

Значение bw	Размер FFT (ns)	Приблизительный RBW
0	4096	~2 500 Гц
1	2048	~5 000 Гц
2	1024	~10 000 Гц
3	512	~25 000 Гц

3.4 **Режим сканирования**

Значение	Название	Описание
0	FAST_MODE	Быстрая развёртка, пониженный динамический диапазон
1	PRECISION_MODE	Увеличенное усреднение
2	TURBO_MODE	Максимальная скорость

4. **Справочник команд**

Команды информации об устройстве

API_GET_DEV_TYPE — 0x1D

Возвращает 16-битный идентификатор типа устройства.

Payload **запроса**: отсутствует (N = 0)

Payload **ответа**: 2 байта

Смещение	Размер	Поле	
0	2	device_type	(uint16 little-endian)

Известные значения типа устройства:

- `0x000A` — Приложение, SigHunt

- `0x000C` — Приложение, SigHunt TG (с трекинг-генератором)

API_GET_DEV_ID — 0x2D

Возвращает уникальный 96-битный аппаратный идентификатор МК (три 32-битных слова).

Payload **запроса**: отсутствует

Payload **ответа**: 12 байт

Смещение	Размер	Поле
0	4	UID[0] (uint32 little-endian)
4	4	UID[1]
8	4	UID[2]

API_GET_SW_VERSION — 0x3D

Возвращает семантическую версию прошивки.

Payload **запроса**: отсутствует

Payload **ответа**: 3 байта

Смещение	Размер	Поле
0	1	VER_MAJOR
1	1	VER_MINOR
2	1	VER_PATCH

API_GET_MIN_FREQ_HW — 0xE0

Возвращает минимальную аппаратную частоту, доступную для настройки устройства.

Payload **запроса**: отсутствует

Payload **ответа**: 8 байт — *uint64* Гц, little-endian

API_GET_MAX_FREQ_HW — 0xE1

Возвращает максимальную аппаратную частоту.

Payload **запроса**: отсутствует

Payload **ответа**: 8 байт — uint64 Гц, little-endian

API_GET_RF_PORTS — 0xE2

Возвращает ASCII-строку с завершающим нулём, описывающую доступные RF-входы (например, "*RF1\0*" или "*RF1,RF2\0*").

Payload **запроса**: отсутствует

Payload **ответа**: переменный — строка с завершающим нулём

Команды сканирования

API_KEEPALIVE — 0xC0

Эхо — устройство возвращает точно тот же 6-байтовый фрейм, который получило. Используется для проверки соединения.

Payload **запроса**: отсутствует

Payload **ответа**: отсутствует (ответ — эхо фрейма запроса)

API_SCANF — 0xC2 (основная команда сканирования)

Выполняет одиночное сканирование спектра и возвращает все измеренные точки в одном ответе.

Payload **запроса**: 19+ байт

Смещение	Размер	Поле	
0	8	start_freq	(uint64, Гц)
8	8	stop_freq	(uint64, Гц)
16	1	rf_in	(индекс RF-входа, начиная с 0)
17	1	bw	(0–3, см. Раздел 3.3)
18	1	scanmode	(0–2, см. Раздел 3.4)
19	2	pts	(uint16 LE, количество точек, необязательно — по умолчанию 1000)

Правила валидации:

- *start_freq* и *stop_freq* должны быть в пределах аппаратных ограничений
- *start_freq* < *stop_freq*
- *bw* ≤ 3
- *scanmode* ≤ 2 (*TURBO_MODE*)

При ошибке валидации → ответ об ошибке (*0xFC*).

Payload **ответа**: *pts_returned* * 8 байт

Смещение	Размер	Поле
0	pts*4	magnitudes[] (массив float32, дБм)
pts*4	pts*4	frequencies[] (массив float32, МГц)

Пример ответа для 1000 точек:

- Байты 0–3 999: *float32[1000]* значения уровня сигнала (дБм)

- Байты 4 000–7 999: *float32[1000]* значения частоты (МГц)

> **Примечание**: Фактическое количество возвращённых точек (*pts_returned*) может быть меньше запрошенного, если аппаратура не может обеспечить такое разрешение для заданной полосы и ширины RBW.

API_SCAN_F_STREAM — 0xC6 (потоковое сканирование)

Выполняет большое многочастичное сканирование с передачей результатов отдельными пакетами.

> **Ограничение транспорта**: Работает только через USB CDC и WebSocket. HTTP/HTTPS не поддерживается — прошивка вернёт ответ об ошибке, если *transport_context* равен NULL (что имеет место для HTTP-обработчиков).

Payload **запроса**: 23 байта

Смещение	Размер	Поле	
0	8	start_freq	(uint64, Гц)
8	8	stop_freq	(uint64, Гц)
16	4	total_points	(uint32 LE, запрошенное количество точек)
20	1	rf_in	
21	1	bw	
22	1	scanmode	

Последовательность ответов — три типа пакетов в следующем порядке:

Пакет метаданных (первый)

Байт команды = *0xC6*, длина payload = 10:

Смещение	Размер	Поле	
0	2	total_chunks	(uint16 LE, общее количество чанков)
4	4	total_points	(uint32 LE, скорректированное фактическое количество)
8	4	points_per_chunk	(uint32 LE, точек на чанк)

Пакеты данных / чанки (по одному на каждый чанк)

Байт команды = *0xC6*, payload = *4 + chunk_points * 8* байт:

Смещение	Размер	Поле	
0	2	chunk_index	(uint16 LE, начиная с 0)
2	2	points_in_chunk	(uint16 LE)
4	chunk_points*4	magnitudes[]	(float32, дБм)
4+chunk_points*4	chunk_points*4	frequencies[]	(float32, МГц)

Завершающий пакет FIN (**последний**)

Байт команды = *0xC6* , длина payload = 2:

Смещение	Размер	Поле	
0	2	status	(0 = успех)

Правила разбивки на чанки:

- Максимум точек в одном сегменте сканирования: 2000
- Запрошенное количество точек ограничивается физически достижимым максимумом: *span*_Гц / *RBW*_Гц
- Минимум 1000 точек, максимум 500 000 точек

Сигнальный генератор (опциональное оборудование)

API_GEN_CONF — 0xC3

Управляет опциональным трекинг-генератором. Возвращает ошибку, если оборудование ТГ не установлено.

Payload **запроса**: 10 байт

Смещение	Размер	Поле	
0	8	freq	(uint64, Гц) — частота генератора
8	1	state	(0 = выкл, 1 = вкл)
9	1	pow_ind	(0–7, индекс уровня мощности)

Валидация: *freq* в пределах аппаратных ограничений, *pow_ind* ≤ 7.

Payload **ответа**: отсутствует (только ACK)

Настройка автономного сканирования

Устройство имеет 8 слотов автономного сканирования, которые работают независимо и передают данные детектору. Каждый слот настраивается отдельно.

API_SET_STANDALONE_CONF — 0xA3

Записывает конфигурацию одного слота автономного сканирования.

Payload **запроса**: минимум 43 байта

Смещение	Размер	Поле	
0	1	conf_index	(0 до N-1, где N = количество слотов)
1	1	active	(0 = выключен, ненулевое = включён)
2	8	start_freq	(uint64, Гц)
10	8	stop_freq	(uint64, Гц)
18	1	rf_in	
19	1	bw	(0–3)
20	1	scanmode	(0–2)
21	4	ld_time	(uint32 LE, время захвата/паузы в мс, 1–600000)
25	4	pix	(uint32 LE, количество точек сканирования, 1–10000
29	4	det_width_mhz	(float32, ширина пика детектора, МГц)
33	4	snr_threshold	(float32, порог ОСШ, дБ, от -100 до 100)
37	4	pulse_ignore	(uint32 LE, время игнорирования импульса, мс)
41	1	civ_filter	(0 = выкл, ненулевое = фильтр CIV включён)
42	4	max_width_mhz	(float32, максимальная ширина сигнала, МГц, от 0 до 1e6)

Ошибки валидации (любая → *0xFC*):

- *conf_index* ≥ количества слотов
- *packet_len* < 43
- *pix* = 0 или > 10000
- *ld_time* = 0 или > 600000
- *snr_threshold* за пределами [-100, 100]
- *max_width_mhz* за пределами [0, 1e6]
- недопустимый диапазон частот, *bw* > 3 или *scanmode* > 2

Payload **ответа**: отсутствует (ACK)

API_GET_STANDALONE_CONF — 0xA4

Читает конфигурацию одного слота автономного сканирования.

Payload **запроса**: 1 байт

Смещение	Размер	Поле
0	1	conf_index

Payload **ответа**: 45 байт (тот же формат, что и payload SET, начиная с *active*, т.е. без *conf_index*)

Смещение	Размер	Поле	
0	1	active	
1	8	start_freq	(uint64)

9	8	stop_freq	(uint64)
17	1	rf_in	
18	1	bw_index	(0–3, восстановлен из ns)
19	1	scanmode	
20	4	ld_time	(uint32)
24	4	pix	(uint32)
28	4	det_width_mhz	(float32)
32	4	snr_threshold	(float32)
36	4	pulse_ignore	(uint32)
40	1	civ_filter	
41	4	max_width_mhz	(float32)

API_SET_ALERT_FLITER — 0xA7

Устанавливает глобальный тип фильтра оповещений, применяемый автономным детектором.

Payload **запроса**: 1 байт

Смещение	Размер	Поле	
0	1	filter_type	(перечисление, специфичное для устройства)

Payload **ответа**: отсутствует (ACK)

API_GET_ALERT_FILTER — 0xA8

Возвращает текущую настройку фильтра оповещений.

Payload **запроса**: отсутствует

Payload **ответа**: 1 байт

Смещение	Размер	Поле
0	1	filter_type

Чтение сигналов детектора

Детектор непрерывно отслеживает слоты автономного сканирования и помечает сигналы, превышающие настроенный порог ОСШ.

API_GET_DETECTOR_SIGNALS — 0xD1

Возвращает текущие активные (тревожные) сигналы — компактная форма.

Payload **запроса**: отсутствует

Payload **ответа**: переменный — массив записей сигналов, каждая по 16 байт

Запись сигнала (16 байт):

Смещение	Размер	Поле		
0	4	f_center	(float32,	МГц — центр сигнала)
4	4	amp	(float32,	дБм)
8	4	score	(float32,	оценка типа/классификатора сигнала)
12	4	rf_index	(uint32 —	RF-вход, на котором обнаружен сигнал)

Общий payload = *signal_count* * 16 байт. Пустой payload означает отсутствие активных оповещений.

API_GET_DETECTOR_SIGNALS_FULL — 0xD4

Возвращает активные сигналы с полной информацией о полосе и времени.

Payload **запроса**: отсутствует

Payload **ответа**: переменный — массив 24-байтовых записей

Запись сигнала (24 байта):

Смещение	Размер	Поле		
0	4	f1	(float32,	МГц — нижняя граница)
4	4	f2	(float32,	МГц — верхняя граница)
8	4	amp	(float32,	дБм)
12	4	score	(float32)	
16	4	rf_index	(uint32)	
20	4	lifetime	(uint32, миллисекунды с момента первого обнаружения)	

Управление списком игнорируемых сигналов

Сигналы можно добавить в список игнорируемых, чтобы детектор не выдавал по ним оповещений.

API_ADD_IGNORE_SIGNAL — 0xE3

Добавляет сигнал в постоянный список игнорируемых.

Payload **запроса**: минимум 16 байт

Смещение	Размер	Поле	
0	4	f1	(float32, МГц)
4	4	f2	(float32, МГц)
8	4	amp	(float32, дБм)
12	4	type	(float32, тип/оценка сигнала)

Payload **ответа**: отсутствует (ACK)

API_REMOVE_IGNORE_SIGNAL — 0xE4

Удаляет сигнал из списка игнорируемых (поиск по f1, f2, amp, type).

Payload **запроса**: 16 байт — тот же формат, что и *API_ADD_IGNORE_SIGNAL*

Payload **ответа**: отсутствует (ACK)

API_GET_IGNORED_SIGNALS — 0xE5

Возвращает все сигналы, находящиеся в списке игнорируемых.

Payload **запроса**: отсутствует

Payload **ответа**: переменный — массив структур *signal_t* (по 44 байта каждая)

> *signal_t* — внутренняя структура. Первые 16 байт соответствуют полям f1, f2, amp, type в формате float32.

API_START_IGNORE_COLLECTION — 0xE6

Инструктирует устройство автоматически собирать и добавлять обнаруженные сигналы в список игнорируемых в течение указанного времени.

Payload **запроса**: 4 байта

Смещение	Размер	Поле	
0	4	duration_ms	(uint32 LE, длительность в мс)

Payload **ответа**: отсутствует (ACK)

API_GET_IGNORE_COLLECTION_STATUS — 0xE7

Возвращает, активна ли в данный момент сессия сбора игнорируемых сигналов.

Payload **запроса**: отсутствует

Payload **ответа**: 1 байт

Смещение	Размер	Поле	
0	1	status	(0 = ожидание, ненулевое = идёт сбор)

Блокировка сканирования

API_LOCK_SCAN — 0xF2

Блокирует или разблокирует движок сканирования. При блокировке RF-усилители отключаются.

Payload **запроса**: 1 байт

Смещение	Размер	Поле	
0	1	lock	(0 = разблокировать, ненулевое = заблокировать)

Payload **ответа**: отсутствует (ACK)

API_GET_LOCK_STATUS — 0xF3

Возвращает текущее состояние блокировки сканирования.

Payload **запроса**: отсутствует

Payload **ответа**: 1 байт

Смещение	Размер	Поле	
0	1	locked	(0 = разблокировано, 1 = заблокировано)

Сетевая конфигурация

API_SET_LWIP_CONF — 0xA5

Настраивает параметры IP-адреса устройства. Изменения вступают в силу немедленно исохраняются во Flash.

Payload **запроса**: *sizeof(lwip_ip_config_t)* байт (49 байт)

Смещение	Размер	Поле		
0	1	use_dhcp	(bool, 0 = статический,	1 = DHCP)
1	16	ip	(ASCII-строка, например	"192.168.1.10\0")
17	16	netmask	(ASCII-строка, например	"255.255.255.0\0")
33	16	gw	(ASCII-строка, например	"192.168.1.1\0")

Payload **ответа**: отсутствует (ACK)

API_GET_LWIP_CONF — 0xA6

Возвращает текущую сетевую конфигурацию.

Payload **запроса**: отсутствует

Payload **ответа**: 49 байт — тот же формат, что и payload *API_SET_LWIP_CONF*

Опциональные / аппаратно-зависимые команды

API_SET_RELAY_STATE — 0xF1 (только сборки с RELAY_OUT)

Управляет опциональным реле.

Payload **запроса**: 2 байта

Смещение	Размер	Поле	
0	1	relay	(0 = разомкнуто, ненулевое = замкнуто)
1	1	block	(ненулевое = также заблокировать RF-вход в прошивке)

Payload **ответа**: отсутствует (ACK)

API_TRACK — *0xC4* (заглушка — не реализована)

Режим трекинга с помощью трекинг-генератора. В настоящее время полностью закомментирован.

5. Ответ об ошибке

Любая команда, не прошедшая валидацию, возвращает:

BB FC 00 00 CKA CKB

Где *CK_A/CK_B* вычисляются по 4 байтам заголовка *{0xBB, 0xFC, 0x00, 0x00}*.

Причины:

- Частота за пределами аппаратных ограничений
- *start_freq* >= *stop_freq*
- *bw* > 3
- *scanmode* > *TURBO_MODE* (2)
- Нарушение граничных условий параметров (например, *pix* > 10000)
- Неизвестная команда (ветка *default*:)
- Несовпадение контрольной суммы (прошивка внутренне устанавливает *command* = *0xFC*)
- Отсутствие опционального оборудования (например, ТГ не установлен для *API_GEN_CONF*)
- *API_SCAN_F_STREAM* вызван без допустимого контекста потокового транспорта

6. Python-**клиент** (USB CDC)

Требуется pyserial: *pip install pyserial*

Замените *PORT* на путь к вашему порту:

- Windows: "*COM3*", "*COM4*" , ...
- Linux: "*/dev/ttyUSB0*", "*/dev/ttyACM0*" , ...
- macOS: "*/dev/tty.usbmodem**", "*/dev/cu.usbmodem**", ...

#!/usr/bin/env python3 """
Arinst R3E — пример клиента Alt API через USB CDC.

Выполняет сканирование спектра (*API_SCANF, 0xC2*) и выводит результаты.

Использование:

```
    pip install pyserial python
    r3e_scan.py
"""

import serial im-
port struct im-
port time

# — Настройки ——————————————————————————————————————————
PORT         = "COM3"              # <-- замените на ваш последовательный порт
BAUDRATE = 115_200
TIMEOUT  = 10                      # секунды ожидания ответа на сканирование

# Параметры сканирования
F_START_HZ    = 100_000_000        # 100 МГц
F_STOP_HZ     = 200_000_000        # 200 МГц
RF_IN         = 0                  # индекс RF-входа (начиная с 0)
BW            = 0                  # индекс полосы: 0=~2.5кГц, 1=~5кГц, 2=~10кГц, 3=~25кГц
SCAN_MODE     = 0                  # 0=FAST, 1=PRECISION, 2=TURBO
PTS           = 1000               # запрошенное количество точек
# ——————————————————————————————————————————————————————

START_BYTE       =  0xBB
CMD_KEEPALIVE  =  0xC0
CMD_SCANF       = 0xC2

def ubx_checksum(data: bytes) -> tuple[int, int]:
    """Накопительная контрольная сумма UBX по всем переданным байтам."""
    ck_a = 0
    ck_b = 0
    for b in data:
        ck_a = (ck_a + b) & 0xFF
        ck_b = (ck_b + ck_a) & 0xFF
    return ck_a, ck_b

def build_frame(cmd: int, payload: bytes = b"") -> bytes:
"""Оборачивает команду и payload в полный фрейм Alt API."""
    header = bytes([START_BYTE, cmd, len(payload) & 0xFF, (len(payload) >> 8) & 0xF)
    body = header + payload
    ck_a, ck_b = ubx_checksum(body)
    return body + bytes([ck_a, ck_b])
```

```
def read_frame(ser: serial.Serial) -> dict:
    """
    Считывает один полный фрейм Alt API из последовательного порта.
    Блокируется до обнаружения стартового байта, затем читает остаток фрейма.
    """
    # Синхронизация со стартовым байтом
    while True:
        b = ser.read(1) if not
        b:
            raise TimeoutError("Таймаут ожидания стартового байта фрейма") if b[0] ==
        START_BYTE:
            break

    # Чтение команды и длины
    header_rest = ser.read(3) if
    len(header_rest) < 3:
        raise TimeoutError("Таймаут чтения заголовка фрейма")

    cmd        = header_rest[0]
    length = header_rest[1] | (header_rest[2] << 8)

    # Чтение payload + 2 байта контрольной суммы
    rest = ser.read(length + 2) if len(rest)
    < length + 2:
        raise TimeoutError("Таймаут чтения payload фрейма")

    payload   = rest[:length] recv_cka =
    rest[length] recv_ckb = rest[length + 1]
    # Проверка контрольной суммы (пропускается если оба байта нулевые — тестовый режим).
    if (recv_cka, recv_ckb) != (0, 0):
        frame_so_far = bytes([START_BYTE, cmd, length & 0xFF, (length >> 8) & 0xFF]
        calc_cka, calc_ckb = ubx_checksum(frame_so_far)
        if (recv_cka, recv_ckb) != (calc_cka, calc_ckb):
        raise ValueError(
                f"Несовпадение контрольной суммы: получено ({recv_cka:#04x},{recv_  ckb:#04x}), "
                 f"ожидалось ({calc_cka:#04x},{calc_ckb:#04x})"
            )

    return {"cmd": cmd, "payload": payload}

def send_keepalive(ser: serial.Serial) -> bool:
    """Отправляет keepalive и проверяет эхо-ответ. Возвращает True при успехе."""
    frame = build_frame(CMD_KEEPALIVE)
```

```
    ser.write(frame)
    ser.flush()
    try:
        resp = read_frame(ser)
        return resp["cmd"] == CMD_KEEPALIVE
    except (TimeoutError, ValueError):
        return False

def scan(ser: serial.Serial,
         f1_hz: int, f2_hz: int,
         rf_in: int = 0, bw: int = 0, mode: int = 0,
         pts: int = 1000) -> tuple[list[float], list[float]]:
    """
    Отправляет API_SCANF и возвращает (уровни_дБм, частоты_МГц).
    """
    payload = struct.pack("<QQBBBH", f1_hz, f2_hz, rf_in, bw, mode, pts)
    frame   = build_frame(CMD_SCANF, payload)

    ser.write(frame)
    ser.flush()

    resp = read_frame(ser)

    if resp["cmd"] == 0xFC:
        raise RuntimeError("Устройство вернуло ошибку (0xFC) — проверьте параметры сканирования")

    if resp["cmd"] != CMD_SCANF:
        raise RuntimeError(f"Неожиданная команда ответа: {resp['cmd']:#04x}")

    data = resp["payload"]
    n    = len(data) // 8                    # каждая точка = 4 байта уровень + 4 байта часто
    mags  = list(struct.unpack_from(f"<{n}f", data, 0))
    freqs  = list(struct.unpack_from(f"<{n}f", data, n * 4))
    return mags, freqs

def main():
    print(f"Подключение к {PORT} на скорости {BAUDRATE} бод ...")
    with serial.Serial(PORT, BAUDRATE, timeout=TIMEOUT) as ser:
        time.sleep(0.1)          # ожидание инициализации USB CDC
        ser.reset_input_buffer()

        print("Отправка keepalive ...")
```

```
    if not send_keepalive(ser):
        print("ПРЕДУПРЕЖДЕНИЕ: нет эхо-ответа на keepalive — устройство может быть не готово ")

    print(f"Сканирование {F_START_HZ/1e6:.1f}–{F_STOP_HZ/1e6:.1f} МГц "
          f"({PTS} точек, bw={BW}, режим={SCAN_MODE}) ...")

    mags, freqs = scan(ser, F_START_HZ, F_STOP_HZ,
                       rf_in=RF_IN, bw=BW, mode=SCAN_MODE, pts=PTS)

    print(f"Получено {len(freqs)} точек")
    print(f"Диапазон частот : {freqs[0]:.3f} – {freqs[-1]:.3f} МГц")
    print(f"Амплитуда       : мин={min(mags):.1f} дБм, макс={max(mags):.1f} дБм

    # Опционально: сохранение в CSV
    csv_path = "scan_result.csv" with
    open(csv_path, "w") as f:
        f.write("freq_mhz,amp_dbm\n")
        for freq, amp in zip(freqs, mags):
            f.write(f"{freq:.4f},{amp:.2f}\n")
    print(f"Сохранено в {csv_path}")

if __name__ == "__main__":
    main()
```

7. **Таблица быстрого справочника**

Команда	ID	Направление	Payload (запрос)	Payload ответа
API_KEEPALIVE	0xC0	R	нет	нет (эхо)
API_SCANF	0xC2	R	1U–21 Б	pts×8 Б (массивы float: уровень+частота)
API_SCAN_F_STREAM	0xC6	R	23 Б	Мультипакет: метаданные + чанки + FIN *(только USB WS)*
API_GEN_CONF	0xC3	W	10 Б	нет
API_GET_DEV_TYPE	0x1D	R	нет	2 Б
API_GET_DEV_ID	0x2D	R	нет	12 Б
API_GET_SW_VERSION	0x3D	R	нет	3 Б

API_GET_MIN_FREQ_HW	0xE0	R	нет	8 Б (uint64))
API_GET_MAX_FREQ_HW	0xE1	R	нет	8 Б (uint64))
API_GET_RF_PORTS	0xE2	R	нет	переменный (ASCII)
API_SET_STANDALONE_CONF	0xA3	W	43 Б	нет
API_GET_STANDALONE_CONF	0xA4	R	1 Б	45 Б
API_SET_ALERT_FLITER	0xA7	W	1 Б	нет
API_GET_ALERT_FILTER	0xA8	R	нет	1 Б
API_SET_LWIP_CONF	0xA5	W	4U Б	нет
API_GET_LWIP_CONF	0xA6	R	нет	49 Б
API_LOCK_SCAN	0xF2	W	1 Б	нет
API_GET_LOCK_STATUS	0xF3	R	нет	1 Б
API_GET_DETECTOR_SIGNALS	0xD1	R	нет	N×16 Б
API_GET_DETECTOR_SIGNALS_FULL	0xD4	R	нет	N×24 Б
API_ADD_IGNORE_SIGNAL	0xE3	W	1G Б	нет
API_REMOVE_IGNORE_SIGNAL	0xE4	W	1G Б	нет
API_GET_IGNORED_SIGNALS	0xE5	R	нет	N×44 Б
API_START_IGNORE_COLLECTION	0xE6	W	4 Б	нет
+API_GET_IGNORE_COLLECTION_ STATUS	0xE7	R	нет	1 Б
API_SET_RELAY_STATE	0xF1	W	2 Б	нет *(опц. Оборудование)*
API_ERR	0xFC	*	—	ответ об ошибке