АртТехЛабс
ArtTechLabs

Руководство по SDK

DOmini SDK



DOmini
ArtTechLabs

Kotlin · Java · Python · C# · C · JavaScript · Swift

arttechlabs@ya.ru
arttechlabs.ru

Руководство описывает работу с осциллографами DOmini и DOmini PRO из прикладных программ: получение кадров, измерения, запись, логический анализ, декодирование протоколов, обмен по шинам и работу с модулями. Сборка и подключение SDK для каждой платформы (macOS, Linux, Windows, Android, браузер, iOS) описаны в документации к соответствующей привязке в каталоге `bindings`.

Примеры приведены на **Kotlin** (полный API) и **Python** (через C API). В Java, C#, C, JavaScript и Swift используются те же вызовы под другими именами; соответствие приведено в последнем разделе. Все примеры руководства проверяются автоматическими тестами на виртуальном приборе: Kotlin — `domini-sdk/src/jvmTest/.../UserGuideTest.kt`, Python — `bindings/python/tests/test_user_guide.py`. Версии в форматах PDF и HTML собираются командой `python3 scripts/build-pdf.py docs/sdk/user-guide.md`.

Содержание

1. Прибор: входы и выходы 3
2. Первая программа 4
3. Кадр осциллографа 5
4. Измерения 8
5. Непрерывный сбор кадров 9
6. Длительная запись: самописец 10
7. Логический анализатор и протоколы 11
8. ШИМ и цифровые выходы 13
9. Шины: SPI, UART, I²C, 1-Wire 14
10. Модули 16
11. Ошибки, таймауты, отключение 17
12. Многопоточность и пользовательский интерфейс 18
13. DOmini или DOmini PRO 19
14. Соответствие API по языкам 20

1. Прибор: входы и выходы

Вывод	Назначение
A, **B**	аналоговые входы осциллографа и самописца
D1…D4	цифровые линии: входы логического анализатора либо выходы и шины (прибор — ведущий)
D1	внешний триггер осциллографа (`TriggerSource.External`); SPI SCK
D2	SPI SDI (MISO); UART RTS
D3	**ШИМ**; SPI CS; UART TX; I²C SDA
D4	SPI SDO (MOSI); UART RX; I²C SCL; 1-Wire (DS18B20)

В каждый момент времени линия выполняет одну функцию. Перед работой логического анализатора линии порта переводятся в режим входов (`io.release()`), а генератор ШИМ останавливается (`pwm.stop()`).

Прибор может использоваться только одной программой. Если прибор открыт в приложении DOmini, во вкладке браузера или в другом процессе, SDK не получит к нему доступ.

2. Первая программа

Отладку рекомендуется начинать на **виртуальном приборе**. Он отвечает теми же байтами протокола, что и настоящий, и выдаёт известные сигналы: A — синусоида 1 кГц ±1 В, B — прямоугольный сигнал 500 Гц 0…3 В с заполнением 30 %, D1…D4 — меандры 1 кГц, 500, 250 и 125 Гц. Код, отлаженный на виртуальном приборе, работает с настоящим без изменений.

```
import com.arttech.device.domini.device.protocol.DeviceModel
import com.arttech.device.domini.sdk.*
import kotlinx.coroutines.runBlocking
import kotlinx.coroutines.withTimeoutOrNull
import kotlin.time.Duration.Companion.seconds

fun main() = runBlocking {
    // Android: DominiConnector.android(context), браузер: .web(), Kotlin/Native: .native()
    val connector = DominiConnector.desktop()
    // Прибор по USB, а если его нет 5 с — виртуальный DOmini PRO.
    val domini = withTimeoutOrNull(5.seconds) { connector.awaitDevice() }
        ?: Domini.demo(DeviceModel.DominiPro)
    println("${domini.model.title} ${domini.serial}, прошивка ${domini.firmwareVersion}")
    println("виртуальный: ${domini.isDemo}")

    val wf = domini.scope.capture(ScopeConfig(channels = setOf(Channel.A, Channel.B)))
    println(wf.measure(Channel.A))

    domini.close()
    connector.close()                                  // освобождает USB
}
```

```
import domini

try:
    d = domini.open_usb(timeout_ms=5000)
except domini.DominiError:
    d = domini.open_demo(pro=True)
with d:                                                # выход из with закрывает прибор
    print(d)          # Domini(DOmini PRO, demo, serial DE30-0000-0002, firmware 2.00)
    wf = d.capture(channels="AB")
    print(wf.measure("A"))
```

`DominiConnector` отслеживает подключение USB-устройств. При отключении прибор закрывается, при новом подключении создаётся новый объект `Domini` (раздел 11). В C, Python и C# коннектор отсутствует: `open_usb` ожидает первый прибор в течение заданного времени.

3. Кадр осциллографа

Кадр — однократное считывание буфера прибора. Основные параметры:

- `channels` — A, B или оба канала.
- `sampleRateHz` — частота АЦП **на все включённые каналы**; при двух каналах на каждый приходится половина. Используется ближайшее значение из `domini.capabilities.oscilloscopeRatesHz`.
- `samples` — число отсчётов на канал; по умолчанию используется наибольший буфер. У DOmini это 13 200 отсчётов на один канал и 6 600 на два (у платы второй ревизии — 26 400 и 13 200), у PRO — до 16K.
- **Длительность кадра** равна числу отсчётов, делённому на частоту одного канала. Например, DOmini, A+B, 1 МГц: 6 600 ÷ 500 кГц ≈ 13 мс.

С ростом частоты длительность кадра уменьшается. Для сигнала 1 кГц подходит частота 1 МГц: кадр содержит около 10 периодов.

```
// DOmini: [1000000, 500000, …]; у PRO — до 20 МГц
println(domini.capabilities.oscilloscopeRatesHz)
val wf = domini.scope.capture(
    ScopeConfig(
        channels = setOf(Channel.A, Channel.B),
        sampleRateHz = 1_000_000,
        trigger = Trigger.rising(TriggerSource.A, levelVolts = 0.5, mode = TriggerMode.Auto),
    ),
)
val a: FloatArray = wf[Channel.A]                    // вольты; wf.timeMs — время, мс
println("${wf.size} отсчётов, ${wf.sampleRateHz / 1000} кГц на канал")
println("триггер: ${wf.triggered}")
```

```
wf = d.capture(channels="AB", sample_rate_hz=1_000_000,
               trigger=domini.Trigger("A", level=0.5, mode="auto"))
# wf.time_ms, wf.a, wf.b — numpy или array('f')
print(len(wf), wf.sample_rate_hz, wf.triggered)
```

Триггер

Без триггера момент начала кадра произволен, и изображение сигнала от кадра к кадру смещается. При использовании триггера нулю времени (`timeMs[i] = 0`) соответствует момент пересечения сигналом заданного уровня в заданном направлении, и кадры совмещаются.

Параметр	Назначение
`source`	канал A, B или внешний вход D1 (`External`, уровень не задаётся)
`edge`	фронт (`Rising`) или спад (`Falling`)
`levelVolts`	уровень, В на входе щупа
`mode`	**`Auto`** — кадр формируется и без события; **`Normal`** — прибор ожидает события без ограничения времени
`pretriggerPercent`	доля кадра до события, 0…90 % (по умолчанию 50)
`hysteresisMv`	порог срабатывания прибора для защиты от ложных срабатываний на шуме (по умолчанию 100 мВ)

Режим `Normal` применяется для регистрации редких событий. Время ожидания следует ограничивать: отмена останавливает прибор, после чего он готов к следующей команде.

```
val event = withTimeoutOrNull(2.seconds) {
    // Trigger.rising без mode — Normal: прибор ждёт события
    domini.scope.capture(ScopeConfig(trigger = Trigger.rising(TriggerSource.A, levelVolts = 0.5)))
}
if (event == null) println("За 2 с сигнал не поднялся выше 0,5 В")
```

```
try:
    wf = d.capture(trigger=domini.Trigger("A", level=0.5), timeout_ms=2000)   # mode="normal"
except domini.DominiTimeout:
    print("За 2 с сигнал не поднялся выше 0,5 В")
```

Вход канала

- **`probe`** — коэффициент деления щупа: 1, 10 или 100. Напряжения пересчитываются на вход щупа. У DOmini в диапазоне –V..+V щуп канала B совпадает со щупом канала A, поскольку смещение нуля у прибора общее для обоих каналов.
- **`coupling = AC`** — без постоянной составляющей: из кадра вычитается середина размаха, уровень триггера отсчитывается от неё.
- **`inputOhms`** — сопротивление внешнего делителя перед входом, Ом.
- **`proRange`** (только PRO) — ±5 В (`Low`) или ±20 В (`High`).
- **`unipolar`** (только DOmini) — диапазон 0..+V вместо –V..+V; обеспечивает более высокое разрешение для сигналов положительной полярности.

```
// Probe — com.arttech.device.domini.device.protocol.Probe
val config = ScopeConfig(a = ChannelInput(probe = Probe.X10, coupling = Coupling.AC))
```

```
wf = d.capture(a=domini.Input(probe=10, ac=True))
```

4. Измерения

`wf.measure(channel)` возвращает минимум, максимум, размах, среднее и среднеквадратичное значения, частоту, период и коэффициент заполнения. Если кадр содержит менее полутора периодов, частота, период и заполнение равны 0; в этом случае уменьшают частоту выборки или увеличивают длину кадра.

`noiseVolts` (по умолчанию 0,1 В) — гистерезис при подсчёте периодов: пересечение среднего значения засчитывается, если сигнал отклоняется от него больше чем на эту величину. Для зашумлённых сигналов значение увеличивают, чтобы устранить нестабильность частоты. Для сигналов с размахом менее 0,2 В значение уменьшают, иначе частота будет равна 0.

```
val m = wf.measure(Channel.A, noiseVolts = 0.05)
println("размах ${m.peakToPeak} В, RMS ${m.rms} В")
println("${m.frequencyHz} Гц, заполнение ${m.dutyPercent} %")
```

```
m = wf.measure("A", noise_volts=0.05)
print(m.peak_to_peak, m.rms, m.frequency_hz, m.duty_percent)
# Любой свой массив: domini.measure(values, time_ms, noise_volts)
```

5. Непрерывный сбор кадров

Для отображения сигнала в реальном времени используется поток кадров: прибор собирает следующий кадр во время обработки текущего. Сбор завершается отменой, при этом прибор останавливается автоматически.

```
domini.scope.frames(config).take(100).collect { wf -> draw(wf) }   // 100 кадров
// До отмены: val job = scope.launch { domini.scope.frames(config).collect(::draw) } … job.cancel()
```

```
for _ in range(100):                                  # в Python — capture в цикле
    draw(d.capture(channels="AB", trigger=domini.Trigger("A", level=0.5, mode="auto")))
```

Java — `streamScope(config, onData, onError)` → `Subscription.close()`; JavaScript — `for await (const wf of domini.scope.frames())` и `break`.

6. Длительная запись: самописец

Длительность кадра ограничена буфером прибора. **Самописец** (потоковый режим) передаёт отсчёты непрерывно и предназначен для записи продолжительностью от минут до часов. Частоты выборки ниже, чем у осциллографа: `capabilities.recorderRatesHz` (DOmini 5…250 кГц, PRO 5…100 кГц). Данные поступают фрагментами длительностью 50–150 мс. Время фрагмента отсчитывается в миллисекундах от начала записи и непрерывно между фрагментами, без пропусков.

```
File("record.csv").bufferedWriter().use { out ->
    out.appendLine("time_ms,a_volts")
    withTimeoutOrNull(10.seconds) {                  // 10 секунд записи
        domini.scope.record(RecorderConfig(sampleRateHz = 10_000)).collect { chunk ->
            val a = chunk[Channel.A]
            for (i in 0 until chunk.size) out.appendLine("${chunk.timeMs[i]},${a[i]}")
        }
    }
}
```

```
import csv, time

with open("record.csv", "w", newline="") as f, d.record(channels="A", sample_rate_hz=10_000) as rec:
    out = csv.writer(f)
    out.writerow(["time_ms", "a_volts"])
    end = time.monotonic() + 10
    for chunk in rec:                                # выход из with останавливает прибор
        out.writerows(zip(chunk.time_ms, chunk.a))
        if time.monotonic() > end:
            break
    print("потеряно кусков:", rec.dropped)           # если обработка не успевала
```

Во время записи остальные вызовы прибора ожидают её завершения. Обработка данных должна успевать за потоком: буфер между прибором и прикладным кодом рассчитан на несколько секунд, при более длительном отставании данные теряются. Ресурсоёмкую обработку следует выполнять в отдельном потоке.

Логический анализатор также поддерживает режим самописца: `domini.logic.record(LogicRecorderConfig(sampleRateHz))`, в Python — `d.record(logic=True)`.

7. Логический анализатор и протоколы

Кадр логического анализатора содержит уровни линий D1…D4. Допустимые частоты — `capabilities.logicRatesHz`. Триггер — фронт или спад на одной из линий. Параметр `pull` задаёт подтяжку входов: `PullUp` — для шин с открытым коллектором (I²C, 1-Wire), в остальных случаях `Disabled`.

```
val capture = domini.logic.capture(
    LogicConfig(
        sampleRateHz = 1_000_000,
        // D3, спад — старт-бит UART
        trigger = LogicTrigger(line = 2, edge = Edge.Falling, mode = TriggerMode.Auto),
        pull = PortPull.PullUp,               // com.arttech.device.domini.device.PortPull
    ),
)
val d1: BooleanArray = capture.lines[0]
println("D1: ${capture.measure(0).frequencyHz} Гц")
```

```
cap = d.capture_logic(sample_rate_hz=1_000_000, pull="up",
                      trigger=domini.LogicTrigger(line=2, edge="falling", mode="auto"))
# line(i) — список 0/1; lines — байт на отсчёт, D1..D4 в битах 0..3
print(cap.measure(0).frequency_hz, cap.line(0)[:16])
```

Декодеры протоколов

В Kotlin, Java и JavaScript кадр логического анализатора поддерживает декодирование протоколов. Назначение линий совпадает с назначением при работе с шинами прибора:

Декодер	Линии
`decodeUart(line, baud, parity, stopBits)`	любая (`line` 0…3), 8 бит данных
`decodeSpi(mode, wordBits)`	D1 SCK, D2 SDI, D3 CS (активный низкий), D4 SDO
`decodeI2c()`	D3 SDA, D4 SCL
`decodeOneWire()`	D4

```
for (w in capture.decodeUart(line = 2, baud = 9600)) {
    // w.error — Framing / Parity или null
    println("%.3f мс: 0x%02X %s".format(w.startMs, w.value, w.error ?: ""))
}
```

Частота выборки должна обеспечивать не менее 10 отсчётов на бит: для 9600 бод — не ниже 100 кГц. В C, Python и C# декодеры отсутствуют; доступны только уровни линий.

8. ШИМ и цифровые выходы

ШИМ формируется на выводе D3 в диапазоне частот от `capabilities.pwmMinHz` до 10 МГц с коэффициентом заполнения 0…100 %. Прибор получает частоту делением на целое число, поэтому `start` возвращает фактически установленную частоту.

Для проверки подключения выход D3 соединяется перемычкой со входом A:

```
val actual = domini.pwm.start(frequencyHz = 1_000, dutyPercent = 25)
val trigger = Trigger.rising(TriggerSource.A, levelVolts = 1.0, mode = TriggerMode.Auto)
val m = domini.scope.capture(ScopeConfig(trigger = trigger)).measure(Channel.A)
println("ШИМ $actual Гц → на A ${m.frequencyHz} Гц, заполнение ${m.dutyPercent} %")
domini.pwm.stop()
```

```
actual = d.pwm_start(1000, duty_percent=25)
m = d.capture(trigger=domini.Trigger("A", level=1.0, mode="auto")).measure("A")
d.pwm_stop()
```

Цифровые выходы. `setOutputs(outputs, levels)` принимает две 4-битные маски (бит 0 — D1): линии, переводимые в режим выхода, и линии с высоким уровнем среди них. Остальные линии остаются входами.

```
// D1 — высокий, D2 — низкий, D3 и D4 — входы
domini.io.setOutputs(outputs = 0b0011, levels = 0b0001)
domini.io.release()                                  // все линии — входы
```

```
d.set_outputs(0b0011, 0b0001)
d.set_outputs(0, 0)
```

9. Шины: SPI, UART, I²C, 1-Wire

Прибор работает ведущим шины. За один обмен передаётся до 52 байт и принимается до 52 байт. Назначение выводов приведено в разделе 1. Ошибка на шине вызывает исключение `DominiIoException` с кодом прибора:

Код	Значение
1	таймаут на шине
2	ведомое устройство не ответило (устройство отсутствует, нет ACK)
3	ошибка контрольной суммы
7	интерфейс не лицензирован на данном приборе

Если прибор не ответил в течение 10 с, вызывается исключение `DominiTimeoutException`.

I²C: поиск устройств и чтение регистра. Адрес задаётся в 7-битном формате.

```
val bus = domini.io.i2c(speedKhz = 100)
val found = (0x08..0x77).filter { bus.probe(it) }
println(found.joinToString { "0x%02X".format(it) })   // виртуальный прибор: 0x40
val id = bus.read(address = 0x40, count = 2, register = byteArrayOf(0xFE.toByte()))
```

```
found = []
for addr in range(0x08, 0x78):
    try:
        d.i2c_read(addr, 1)
        found.append(addr)
    except domini.DominiIoError as e:
        if e.device_code != 2:                      # 2 — нет ответа по этому адресу
            raise
reg = d.i2c_read(0x40, 2, register=b"\xFE")
```

SPI. Полнодуплексный обмен: число принятых байтов равно числу переданных. Режим 0…3 вычисляется как CPOL·2 + CPHA. Пример — чтение идентификатора флеш-памяти (команда 0x9F):

```
val spi = domini.io.spi(SpiConfig(frequencyHz = 1_000_000, mode = 0))
val rx = spi.transfer(byteArrayOf(0x9F.toByte(), 0, 0, 0))
// rx[1] — производитель, rx[2..3] — тип и объём
```

```
rx = d.spi_transfer(b"\x9F\x00\x00\x00", frequency_hz=1_000_000, mode=0)
```

UART. D3 — линия TX прибора, D4 — RX. `read(count)` ожидает ровно `count` байт в течение `timeoutMs` (по умолчанию 1 с); по истечении времени возвращается ошибка прибора.

```
val port = domini.io.uart(UartConfig(baud = 9600))
port.write("AT\r\n".encodeToByteArray())
val reply = port.read(8).decodeToString()
```

```
d.uart_write(b"AT\r\n", baud=9600)
reply = d.uart_read(8, baud=9600, timeout_ms=1000)
```

1-Wire — вывод D4. Импульсы сброса и присутствия формируются прибором автоматически: `io.oneWire().read(command, count)` / `write(data)`. Для DS18B20 предусмотрен готовый модуль (раздел 10).

10. Модули

Модули ArtTechLabs подключаются к линиям D1…D4. SDK реализует их протоколы и возвращает значения в физических единицах:

```
val celsius = domini.modules.ds18b20(resolution = 12).read()     // DS18B20 на D4, °C
// INA226: Range.Low — ±80 мА, Range.High — ±15 А (Ina226 — device.protocol.modules)
val power = domini.modules.ina226(Ina226Config(range = Ina226.Range.Low)).capture()
println("${power.meanCurrentA} A, ${power.meanVoltageV} В, ${power.meanPowerW} Вт")
val adc = domini.modules.mcp3561().capture()     // 24-битный АЦП: adc.volts, adc.codes
val set = domini.modules.mcp4921().set(1500.0)   // ЦАП, мВ → выставленное, мВ
```

```
celsius = d.temperature(resolution=12)
current_a, voltage_v = d.ina226(high_range=False)
adc = d.adc(domini.ADC_MCP3561)                      # вольты — в adc.a
actual_mv = d.dac(1500)
```

Модуль	Назначение	Примечания
DS18B20	температура	9…12 бит, проверка CRC; `readings()` — поток
INA226	ток, напряжение, мощность	`capture` / `frames` / `record`
MCP3561	АЦП 24 бит	триггер по уровню — начиная с прошивки 1.10
AD7683	АЦП 16 бит, 5…100 кГц	только DOmini
MCP3550	АЦП 22 бит, ~12 отсчётов/с	только поток; только DOmini
MCP4921	ЦАП 0…3000 мВ	

Обращение к модулю, не поддерживаемому моделью прибора, вызывает `UnsupportedOperationException` (в C — код `DOMINI_E_UNSUPPORTED`).

11. Ошибки, таймауты, отключение

Ситуация	Kotlin / Java	Python	Действие
недопустимый параметр	`IllegalArgumentException`	`ValueError`	исправить вызов
прибор не ответил, истекло время ожидания	`DominiTimeoutException`	`DominiTimeout`	повторить вызов; прибор уже остановлен
прибор отключён или закрыт	`DominiDisconnectedException`	`DominiDisconnected`	ожидать повторного подключения
ошибка шины	`DominiIoException(code)`	`DominiIoError.device_code`	см. коды в разделе 9

После любой ошибки прибор остаётся работоспособным: SDK останавливает его и считывает оставшиеся ответы. Повторно открывать прибор не требуется.

Повторное подключение. В Kotlin коннектор сообщает о подключении и отключении прибора. Сбор внутри `collectLatest` перезапускается для каждого нового прибора:

```
connector.start()
connector.state.collectLatest { state ->
    val domini = (state as? ConnectionState.Connected)?.domini ?: return@collectLatest
    try {
        domini.scope.frames(config).collect(::draw)
    } catch (e: DominiDisconnectedException) {
        // кабель выдернули — ждём следующего Connected
    }
}
```

В Python и C# прибор открывается повторно в цикле: `open_usb` ожидает подключения прибора.

12. Многопоточность и пользовательский интерфейс

- **Kotlin.** Все операции — `suspend`-функции. Обмен с прибором выполняется под одним мьютексом, поэтому вызовы из разных корутин выполняются последовательно. Сбор кадров выполняется вне главного потока: `frames(config).flowOn(Dispatchers.Default)`.
- **Android.** Сбор выполняется, пока экран отображается (`repeatOnLifecycle(STARTED)`); в противном случае прибор продолжает работать в фоне. Методы Java-API блокирующие; вызывать их в главном потоке недопустимо (ANR).
- **Java.** Слушатели `stream…` / `record…` вызываются в фоновом потоке SDK. Поток задаётся методом `setCallbackExecutor` (Android — `context.getMainExecutor()`, Swing — `SwingUtilities::invokeLater`).
- **Python, C#, C.** Вызовы блокирующие; вызовы из разных потоков выполняются последовательно. Функция обратного вызова самописца (C#, C, Python с `callback=`) вызывается в фоновом потоке библиотеки; обновлять из неё пользовательский интерфейс напрямую нельзя.
- **JavaScript.** Все операции возвращают обещания (`Promise`) или асинхронные итераторы (`for await`); блокирующих вызовов нет.

13. DOmini или DOmini PRO

Модель прибора определяется автоматически (`domini.model`). Различия моделей отражены в `domini.capabilities`:

	DOmini	DOmini PRO
АЦП	10 бит	12 бит
Осциллограф	до 1 МГц, буфер до 13 200 отсчётов (26 400 у второй ревизии)	до 20 МГц, буфер до 16K
Диапазоны	–V..+V или 0..+V (`unipolar`)	±5 В или ±20 В (`proRange`) на канал
Совмещённый режим (A/B + D1…D4 в одном кадре, `digital = true`)	есть	нет
Самописец	5…250 кГц	5…100 кГц
Модули AD7683, MCP3550	есть	нет

Для совместимости с обеими моделями параметры следует выбирать по `capabilities`, а не по модели:

```
val rate = domini.capabilities.oscilloscopeRatesHz.first()   // наибольшая частота модели
val combined = domini.capabilities.combinedMode              // можно ли digital = true
```

Соответствие API по языкам

Kotlin, Java, JavaScript — фасады SDK на Kotlin:

Действие	Kotlin	Java (`JDomini`)	JavaScript
открыть	`connector.awaitDevice()`, `Domini.demo()`	`DominiDesktop.connect(ms)`, `DominiJava.demo()`	`connector.awaitDevice()`, `Domini.demo()`
кадр	`scope.capture(cfg)`	`captureScope(cfg, ms)`	`scope.capture(opts)`
непрерывный сбор	`scope.frames(cfg)`	`streamScope(cfg, …)`	`scope.frames(opts)`
измерения	`wf.measure(Channel.A)`	`wf.measure(Channel.A)`	`wf.measure("A")`
самописец	`scope.record(cfg)`	`recordScope(cfg, …)`	`scope.record(opts)`
логический анализатор	`logic.capture(cfg)`	`captureLogic(cfg, ms)`	`logic.capture(opts)`
декодеры	`capture.decodeUart(…)`	`capture.decodeUart(…)`	`capture.decodeUart(…)`
ШИМ	`pwm.start(hz, duty)`	`startPwm(hz, duty)`	`pwm.start(hz, duty)`
выходы	`io.setOutputs(o, l)`	`setOutputs(o, l)`	`io.setOutputs(o, l)`
I²C	`io.i2c(k).read(…)`	`i2cRead(k, a, reg, n)`	`io.i2cRead(…)`
DS18B20	`modules.ds18b20().read()`	`readTemperature(12)`	—
закрыть	`close()`	`close()` / try-with-resources	`close()`

C#, Python, C — C API (`libdomini`) и обёртки над ним:

Действие	C#	Python	C
открыть	`Domini.OpenUsb(ms)`, `OpenDemo()`	`open_usb()`, `open_demo()`	`domini_open_usb`, `domini_open_demo`
кадр	`Capture(cfg, ms)`	`capture(...)`	`domini_scope_capture`
непрерывный сбор	`Capture` в цикле	`capture` в цикле	`domini_scope_capture` в цикле
измерения	`wf.Measure(Channels.A)`	`wf.measure("A")`	`domini_measure`
самописец	`StartRecording(…)`	`record(...)`	`domini_record_start`
логический анализатор	`CaptureLogic(…)`	`capture_logic(...)`	`domini_logic_capture`
декодеры	—	—	—
ШИМ	`StartPwm(hz, duty)`	`pwm_start(hz, duty)`	`domini_pwm_start`
выходы	`SetOutputs(o, l)`	`set_outputs(o, l)`	`domini_set_outputs`
I²C	`I2cRead(a, n, reg)`	`i2c_read(a, n, reg)`	`domini_i2c_read`
DS18B20	`ReadTemperature()`	`temperature()`	`domini_ds18b20_read`
закрыть	`Dispose()` / `using`	`close()` / `with`	`domini_close`

Swift на iOS использует пакет `DominiKit` (кадр, поток кадров, логический анализатор; остальные функции — через Kotlin API), Swift на macOS — C API. Полное описание API: Kotlin — KDoc в пакете `com.arttech.device.domini.sdk`, C — заголовочный файл `domini.h`, JavaScript — файл типов `domini-web.d.ts`, Python и C# — docstring и XML-комментарии в исходных кодах обёрток (каталог `bindings`).