

B/C Signals Administrator and Technical Operations Manual

Table of Contents

1. [Introduction](#)
 - [System Architecture Overview](#)
2. [Signal Ingestion & Processing](#)
 - [Feature: Dual Signal Ingestion \(HTTP & TCP\)](#)
 - [Feature: Secure Asynchronous TCP Server](#)
 - [Feature: Centralized Signal Gatekeeping](#)
3. [Configuration & State Management](#)
 - [Feature: Dynamic Configuration](#)
 - [Feature: Global State Management](#)
4. [Administrative Control \(Telegram Commands\)](#)
 - [Feature: Operational Control \(/pause, /resume\)](#)
 - [Feature: Live Configuration \(/set\)](#)
 - [Feature: Broadcast Channel Management \(/chats\)](#)
 - [Feature: Real-Time Monitoring \(/stats\)](#)
 - [Feature: Utility Commands \(/help, /cancel\)](#)
5. [System Resilience & Diagnostics](#)
 - [Feature: Resilient Signal Processing \(QueueService\)](#)
 - [Feature: Automated Failure Reporting \(/reports\)](#)
 - [Feature: Structured JSON Logging](#)

1. Introduction

This manual provides a complete technical overview for administrators of the **B/C Signals** bot. It details the system's architecture, features, and the administrative commands required for successful operation, monitoring, and maintenance.

System Architecture Overview

The system is a sophisticated Python application built on the **FastAPI** framework, designed for high performance and resilience. Its core components are:

- **FastAPI Web Server:** Provides the primary RESTful API for HTTP-based signal ingestion (POST /signal).
- **Asynchronous TCP Server:** A parallel, secure TCP server for low-latency, persistent connections from trading platforms like MQL5.
- **Telegram Bot Interface:** The command and control center for all administrative tasks,

- configuration, and monitoring.
- **SQLite Database:** A local database providing persistence for signals, system state, managed channels, and dynamic settings.
- **Service Layer:** A modular set of services (SignalService, TelegramService, QueueService) that encapsulate all core business logic.
- **Asynchronous Tasking:** Extensive use of asyncio allows for non-blocking I/O to handle HTTP requests, TCP connections, and Telegram updates concurrently and efficiently.

2. Signal Ingestion & Processing

Feature: Dual Signal Ingestion (HTTP & TCP)

- **What It Is/How It Works:** The system receives trading signals through two distinct, concurrent entry points:
 1. **HTTP API:** A POST /signal endpoint provided by the FastAPI web server.
 2. **TCP Server:** A dedicated asynchronous TCP server running on a separate port.
- **What It Does:** It provides two methods for external applications (webhooks, EAs) to send signals to the bot. Both entry points require a secret_key for authentication and feed signals into the same SignalService for processing.
- **Administrator Value:** This dual-architecture provides critical flexibility. You can use the simple, standard HTTP endpoint for web-based services, while leveraging the high-performance, lower-latency TCP server for dedicated, persistent connections from trading terminals (e.g., MT5), which is often preferred for time-sensitive signal transmission.

Feature: Secure Asynchronous TCP Server

- **What It Is/How It Works:** The TCP server is built with asyncio and can be secured with SSL/TLS certificates. It uses a custom length-prefixed JSON protocol, where each message is preceded by a 4-byte integer defining the payload length. It requires an authentication message (containing the secret_key) as the first-ever message from a new client. It also implements a 60-second heartbeat timeout, closing connections that send no data.
- **What It Does:** It provides a secure, encrypted, and robust channel for direct communication with trading terminals. It authenticates clients, handles message framing, and automatically cleans up stale connections.
- **Administrator Value:** This is a mission-critical feature for robust integration. SSL/TLS encryption ensures signal data is confidential and secure from tampering. The authentication and heartbeat mechanisms provide solutions for two common administrative problems: they prevent unauthorized access and protect the server from being overwhelmed by dead or unresponsive client connections.

Feature: Centralized Signal Gatekeeping

- **What It Is/How It Works:** The SignalService contains a can_send_signal method that

acts as the single gatekeeper for *all* signals, regardless of their source (HTTP or TCP). Before any signal is saved or broadcast, it *must* pass a sequence of checks:

1. Is the bot globally active (i.e., not /paused)?
 2. Is the current time within the configured TRADING_START_TIME and TRADING_END_TIME?
 3. Has enough time passed since the last signal (MIN_SECONDS_BETWEEN_SIGNALS)?
 4. Has the daily signal limit (MAX_SIGNALS_PER_DAY) been reached?
- **What It Does:** It strictly enforces all administrative and risk-management rules on every single signal.
 - **Administrator Value:** This feature provides total, centralized control. It solves the problem of signal flooding, prevents unwanted trades outside of defined market hours, and allows you to globally halt the entire system with a single command. This gatekeeper is your primary tool for system control and risk management.

3. Configuration & State Management

Feature: Dynamic Configuration

- **What It Is/How It Works:** On startup, the bot first loads settings from a static .env file. It then immediately connects to the SQLite database and loads any settings from the settings table. These database values *override* the .env values, and the reload_settings_from_db function intelligently casts them to their correct data types (e.g., int, bool, time).
- **What It Does:** It allows for the bot's core parameters (rate limits, trading hours) to be changed in real-time by modifying the database.
- **Administrator Value:** This is a powerful solution for system flexibility and uptime. You can re-configure the bot *without* a server restart or deployment. This is directly linked to the /set command, allowing you to instantly adapt to changing market conditions (e.g., tighten rate limits during high volatility) with zero downtime.

Feature: Global State Management

- **What It Is/How It Works:** The bot's primary operational status is controlled by a single bot_active flag stored in the system_state table in the database.
- **What It Does:** This flag acts as a global "master switch" for signal processing. The /pause and /resume commands simply toggle this one value.
- **Administrator Value:** This provides a simple, robust, and persistent "emergency stop" solution. Because the state is in the database, it persists even if the bot is restarted. You can confidently pause the system for maintenance or during a crisis, knowing it will not resume until you explicitly command it to.

4. Administrative Control (Telegram Commands)

All administrative commands are restricted to users whose Telegram IDs are listed in the

ADMIN_USER_IDS configuration.

Feature: Operational Control (/pause, /resume)

- **What It Is/How It Works:** The /pause command sets the bot_active flag in the system_state table to false. The /resume command sets it to true.
- **What It Does:** This immediately halts or re-enables all new signal processing.
- **Administrator Value:** This is your most direct and critical control. It provides the "emergency stop" needed for maintenance, unexpected market events, or to prevent signals during a known issue.

Feature: Live Configuration (/set)

- **What It Is/How It Works:** An interactive, conversation-based command. It presents a menu of settings: MAX_SIGNALS_PER_DAY, MIN_SECONDS_BETWEEN_SIGNALS, TRADING_START_TIME, and TRADING_END_TIME. It guides the admin to provide a new value, validates the input format, saves it to the settings database table, and then triggers the in-memory reload.
- **What It Does:** Allows an admin to change the bot's core operational parameters in real-time from their phone.
- **Administrator Value:** This feature provides immense adaptive control. It solves the problem of needing to edit configuration files or restart the server to adapt to market conditions. You can dynamically tune the bot's behavior (e.g., widen trading hours, increase signal limits) with no technical intervention or downtime.

Feature: Broadcast Channel Management (/chats)

- **What It Is/How It Works:** An interactive menu to manage the managed_chats database table. It provides three options:
 1. **List Chats:** Shows all configured channels, automatically refreshing their names via the Telegram API.
 2. **Add Chat:** Prompts for a new Chat ID, validates it, and adds it to the broadcast list.
 3. **Remove Chat:** Presents a list of chats to remove (and prevents removing the default chat).
- **What It Does:** Gives administrators full control over where signal alerts are broadcast.
- **Administrator Value:** This is a solution for flexible signal distribution. You can manage multiple subscriber channels, set up test channels, or migrate services without needing to manually edit the database or configuration files. It centralizes all channel management within the Telegram interface.

Feature: Real-Time Monitoring (/stats)

- **What It Is/How It Works:** A command that queries the signals table to calculate and display key metrics for the current day (UTC).
- **What It Does:** It displays a real-time summary, including:
 - Total signals vs. daily limit

- BUY vs. SELL breakdown
- Number of closed trades
- Win/Loss count and total P&L
- Current bot status (Active/Paused)
- **Administrator Value:** This is your primary, at-a-glance "health check" dashboard. It provides an instant solution for monitoring both system status (bot status, signal count) and trading performance (P&L, win rate) from anywhere.

Feature: Utility Commands (/help, /cancel)

- **What It Is/How It Works:** /help displays a summary of available commands. /cancel terminates any active interactive conversation (like /set or /chats).
- **What It Does:** Provides user support and a safety-exit.
- **Administrator Value:** /cancel is an essential utility that prevents an admin from getting "stuck" in a command, ensuring the interface is always usable.

5. System Resilience & Diagnostics

Feature: Resilient Signal Processing (QueueService)

- **What It Is/How It Works:** A background async task that manages an in-memory retry queue. If a signal fails to process (e.g., due to a temporary database lock), it is added to this queue. The worker then attempts to re-process it after a RETRY_DELAY.
- **What It Does:** It provides fault tolerance for signal processing. Instead of failing permanently, a signal is automatically retried.
- **Administrator Value:** This is a critical solution for reliability. It protects against transient, non-fatal errors (like network blips or temporary DB locks) that could otherwise cause lost signals. It ensures that temporary issues do not compromise the integrity of the signal flow.

Feature: Automated Failure Reporting (/reports)

- **What It Is/How It Works:** The QueueService will only retry a signal a MAX_RETRIES number of times or for SIGNAL_EXPIRY_MINUTES. If a signal fails all retries (RETRY_FAILURE) or becomes too old (STALE_SIGNAL), it is discarded. When this happens, the service:
 1. Logs a detailed report to the reports table in the database.
 2. Sends a direct notification to all admins.
 The /reports command allows an admin to view these unread reports, after which they are marked as is_read.
- **What It Does:** It creates a high-visibility, persistent log of *failed* signals and provides an interface to investigate them.
- **Administrator Value:** This system provides the perfect balance of automation and oversight. You aren't spammed for temporary issues, but you are *immediately* notified of a permanently lost signal. The /reports command provides the diagnostic data (the full

signal details) needed to investigate the root cause, solving the problem of "silent failures."

Feature: Structured JSON Logging

- **What It Is/How It Works:** The logging system is configured to output logs in a structured JSON format. Log files are automatically rotated daily. Logs are also sent to the console in a human-readable format.
- **What It Does:** It creates machine-readable, daily log files that are separate from the user-facing diagnostic reports.
- **Administrator Value:** This is a solution for deep, long-term diagnostics. Structured logs are essential for any serious debugging or performance analysis. They can be ingested directly into log management systems (like ELK, Datadog, or Splunk) for powerful searching, filtering, and dashboarding, which is invaluable for troubleshooting complex issues.