B/C Signals Administrator and Technical Operations Manual

Developed by HCX Technologies

Table of Contents

1. Introduction
2. System Architecture Overview
3. Configuration Management
4. Database Schema and Data Persistence
5. Signal Reception Endpoints
6. Secure TCP Server for Direct Integration
7. Telegram Bot Interface
8. Core Service Layer
9. Logging and Diagnostics
10. Administrative Command Reference
11. Reports and System Diagnostics

---

1. Introduction

The B/C Signals trading bot is a sophisticated Python-based signal relay and management system built on the FastAPI framework. This manual provides comprehensive technical documentation for administrators and technical users responsible for deploying, configuring, and maintaining the bot.

---

2. System Architecture Overview

What It Is / How It Works

The system employs a multi-component architecture consisting of:

- **FastAPI Web Server**: RESTful API for HTTP-based signal ingestion
- **Asynchronous TCP Server**: Parallel SSL/TLS-encrypted server for direct MT5 integration

- **Telegram Bot**: Administrative interface and notification broadcaster
- **SQLite Database**: Local persistence layer for signals, state, and configuration
- **Service Layer**: Modular business logic encapsulation (`SignalService`, `TelegramService`, `QueueService`)
- **Asynchronous Task Management**: Python `asyncio` library for concurrent, non-blocking I/O operations

The signal processing flow operates as follows:

1. Signal received via FastAPI endpoint or TCP server
2. Authentication and validation performed
3. `SignalService` applies status, rate-limit, and trading hours checks
4. Valid signals saved to SQLite database
5. `TelegramService` formats and broadcasts alert to managed channels
6. Failed signals enter `QueueService` retry queue
7. Administrators monitor and control via Telegram Bot interface

What It Does

Provides a robust, multi-channel signal ingestion and distribution system with centralized control and monitoring capabilities.

Administrator Value

Solves: The complexity of managing multiple signal sources and distribution channels while maintaining system reliability.

Benefits:

- Eliminates single points of failure through parallel ingestion pathways
- Centralizes monitoring and control through a single Telegram interface
- Enables real-time system management without server access
- Provides automatic failover through queue-based retry mechanisms

---

3. Configuration Management

What It Is / How It Works

Configuration operates in a two-tier hierarchy:

Tier 1 - Initial Load:

- Settings loaded from `.env` file into Pydantic `Settings` model on startup
- Contains essential credentials and security keys

Tier 2 - Dynamic Overrides:

- Database `settings` table queried after initial load
- Values override `.env` settings in memory
- `reload_settings_from_db` function handles intelligent type casting from text storage to Python types (int, bool, datetime.time)

Key Configuration Parameters:

```
TELEGRAM_BOT_TOKEN          # Bot API authentication
TELEGRAM_DEFAULT_CHAT_ID    # Primary notification channel
WEBHOOK_SECRET_KEY          # Signal authentication key
ADMIN_USER_IDS              # Authorized administrator Telegram IDs
MAX_SIGNALS_PER_DAY         # Daily rate limit ceiling
MIN_SECONDS_BETWEEN_SIGNALS # Minimum signal interval
TRADING_START_TIME          # Trading window start (HH:MM)
TRADING_END_TIME            # Trading window end (HH:MM)
TCP_HOST                    # TCP server bind address
TCP_PORT                    # TCP server port
SSL_CERT_PATH               # SSL certificate file path
SSL_KEY_PATH                # SSL private key file path
```

What It Does

Enables zero-downtime configuration changes by separating static credentials from operational parameters and allowing database-backed dynamic reconfiguration.

Administrator Value

Solves: The operational burden of restarting services for parameter adjustments and the risk of downtime during configuration changes.

Benefits:

- Change rate limits and trading hours without service interruption
- Maintain security credentials in version-controlled `.env` files

- Adjust operational parameters through Telegram commands
- Immediate configuration updates reflected across all system components
- Eliminates SSH access requirements for routine adjustments

---

4. Database Schema and Data Persistence

What It Is / How It Works

SQLite database initialized on startup with five-table schema:

1. `signals` Table:

```
id                # Primary key
timestamp         # Signal creation time
action            # BUY, SELL, or CLOSE
symbol            # Trading instrument
price             # Entry/exit price
closed            # Boolean trade closure flag
close_price       # Exit price (for closed trades)
close_timestamp   # Exit time
profit_loss       # Realized P&L
sent_to_telegram  # Notification delivery flag
telegram_message_id # Telegram message reference
```

2. `system_state` Table:

```
key   # State parameter name
value # State parameter value
```

Primary use: `bot_active` flag for global pause/resume control

3. `managed_chats` Table:

```
chat_id   # Telegram chat identifier
chat_name # Human-readable chat name (auto-updated)
```

4. `settings` Table:

```
key   # Setting parameter name
value # Setting parameter value (text, requires casting)
```

5. `reports` Table:

```
id         # Primary key
timestamp   # Report creation time
report_type # STALE_SIGNAL, RETRY_FAILURE, etc.
details     # JSON/text diagnostic payload
is_read     # Admin acknowledgment flag
```

All database interactions encapsulated in `repository.py` data access layer.

What It Does

Provides persistent storage for trading history, system state, configuration overrides, and diagnostic reports with a clean separation between operational data and control parameters.

Administrator Value

Solves: Data integrity concerns, configuration persistence across restarts, and audit trail requirements.

Benefits:

- Complete trading history for compliance and analysis
- Survives system restarts and crashes without data loss
- Enables post-mortem analysis through detailed signal records
- Diagnostic reports capture non-critical failures for review
- Single-file database simplifies backup procedures
- No external database dependencies reduces deployment complexity

---

5. Signal Reception Endpoints

What It Is / How It Works

FastAPI HTTP Endpoints:

`POST /signal`:

1. Validates `secret_key` in request payload (401 on failure)

2. Validates `action` field (must be BUY, SELL, or CLOSE)
3. Invokes `signal_service` processing pipeline
4. Returns `SignalResponse` with status, message, `signal_id`, and daily signal count
5. HTTP status codes: 200 (success), 401 (unauthorized), 429 (rate limited), 400 (validation error)

`GET /health`:

- Returns service status and current `bot_active` state

`GET /stats`:

- Public endpoint exposing daily statistics, bot status, and current rate limits

What It Does

Exposes RESTful HTTP interface for signal ingestion from webhooks, trading platforms, and third-party automation tools.

Administrator Value

Solves: Integration challenges with diverse signal sources that support HTTP but not direct socket connections.

Benefits:

- Standard HTTP interface compatible with TradingView, webhooks, and automation platforms
- Clear authentication model prevents unauthorized signal injection
- Granular error responses enable rapid troubleshooting
- Health check endpoint supports monitoring and load balancer integration
- Public stats endpoint allows dashboard integration without authentication

---

6. Secure TCP Server for Direct Integration

What It Is / How It Works

Dedicated asynchronous TCP server running parallel to FastAPI application:

Security Layer:

- SSL/TLS encryption using `cert.pem` and `key.pem`
- Configured via `SSL_CERT_PATH` and `SSL_KEY_PATH`

Protocol Specification:

- Length-prefixed JSON messaging
- Each message preceded by 4-byte integer (message length)
- Handles message framing over TCP streams

Connection Flow:

1. Client connects to `TCP_HOST:TCP_PORT`
2. First message must contain valid `secret_key` (connection terminated on failure)
3. Server awaits messages with 60-second timeout
4. Client sends periodic `{"type": "ping"}` heartbeats
5. Server responds with `{"type": "pong"}`
6. Stale connections (no message in 60s) automatically closed

Thread Safety:

- All database operations wrapped in `asyncio.to_thread()`
- Prevents blocking of main event loop from synchronous `sqlite3` operations

What It Does

Provides low-latency, persistent, encrypted connection channel for direct integration with MetaTrader 5 Expert Advisors and other trading terminals.

Administrator Value

Solves: The high latency and connection overhead of HTTP polling, and the security concerns of unencrypted signal transmission.

Benefits:

- Sub-second signal delivery for time-sensitive trading operations
- Persistent connections eliminate per-signal connection overhead
- SSL/TLS encryption protects proprietary trading signals in transit
- Heartbeat mechanism automatically detects and cleans up dead connections
- Thread-safe database access prevents corruption from concurrent operations
- Runs independently from HTTP server, providing redundancy

- Ideal for co-located servers requiring maximum speed

---

7. Telegram Bot Interface

What It Is / How It Works

Built on `python-telegram-bot` library with conversation handler architecture:

Authorization Layer:

- All administrative commands verify user ID against `ADMIN_USER_IDS`
- Unauthorized attempts receive "⛔ Unauthorized" response

Command Processing:

- Basic commands: Direct database operations
- Interactive commands: Conversation handlers with inline keyboards
- State management for multi-step interactions
- Input validation with re-prompting on errors

Alert Broadcasting:

- `send_alert` method iterates through `managed_chats` table
- Formatted HTML messages with emojis and real-time statistics
- Individual message IDs tracked in `signals` table

Admin Notifications:

- `notify_admins` sends direct messages to all `ADMIN_USER_IDS`
- Used for critical events (retry failures, stale signals)

What It Does

Serves as the complete administrative control panel and notification distribution system through Telegram's messaging platform.

Administrator Value

Solves: The need for secure remote administration without VPNs or SSH access, and the requirement for immediate notification delivery.

Benefits:

- Manage bot from any device with Telegram (mobile, desktop, web)
- No server access required for routine operations
- Inline keyboards prevent command syntax errors
- Real-time notifications ensure immediate awareness of system events
- Multi-chat broadcasting enables team and subscriber management
- Authorization system prevents unauthorized configuration changes
- Conversation handlers guide admins through complex operations safely

---

8. Core Service Layer

What It Is / How It Works

Business logic separated into three specialized services:

`SignalService`:

Gatekeeper Method - `can_send_signal`:

```
python

1. Check bot_active flag (system_state table)
2. Validate current time within TRADING_START_TIME and TRADING_END_TIME
3. Verify MIN_SECONDS_BETWEEN_SIGNALS elapsed since last signal
4. Confirm daily count below MAX_SIGNALS_PER_DAY
```

Signal Processing:

- `process_new_signal`: BUY/SELL signal handling
- `process_close_signal`: Trade closure and P&L calculation
- Message formatting with `_format_signal_message` and `_format_close_message`

`QueueService`:

Background Worker:

- Asynchronous task consuming retry queue
- Configurable `RETRY_DELAY` between attempts

- `MAX_RETRIES` (5) before permanent failure

Failure Handling:

- Signals older than `SIGNAL_EXPIRY_MINUTES` (3) marked stale and discarded
- Report generation for both stale signals and retry failures
- Admin notifications for all discarded signals

`TelegramService`:

- Command handler registration
- Conversation state management
- Message formatting and delivery
- Chat management operations

What It Does

Encapsulates all business logic with clear separation of concerns, enabling maintainable and testable code through modular service architecture.

Administrator Value

Solves: Code maintainability challenges, debugging complexity, and the difficulty of isolating failures in monolithic systems.

Benefits:

- Clear separation enables targeted troubleshooting
- `SignalService` gatekeeper prevents invalid signals from entering system
- `QueueService` provides automatic recovery from transient failures
- Retry logic with expiration prevents signal staleness
- Report generation creates audit trail of system issues
- Modular design allows service-level monitoring and metrics
- Failure isolation prevents cascade effects across services

---

9. Logging and Diagnostics

What It Is / How It Works

Structured JSON Logging:

- All log entries output in JSON format
- Compatible with ELK stack, Datadog, Splunk, and other log aggregation platforms
- Enables programmatic parsing and filtering

Log Rotation:

- Daily rotation pattern: `bot_YYYY-MM-DD.log`
- Configured in `app.core.logging_config.py`
- Prevents single-file growth and enables time-based log analysis

Dual Output:

- File output: Structured JSON format
- Console output: Human-readable format for development

What It Does

Provides production-grade logging infrastructure with structured output suitable for modern observability platforms and long-term retention.

Administrator Value

Solves: The difficulty of parsing unstructured logs, the challenge of correlating events across components, and the need for automated alerting based on log patterns.

Benefits:

- JSON structure enables automated log analysis and alerting
- Daily rotation simplifies log retention policy implementation
- Searchable format allows rapid incident investigation
- Integration with log aggregation platforms enables centralized monitoring
- Structured fields support advanced queries (filter by signal_id, action, etc.)
- Automatic date-based file naming enables chronological analysis

- Console output maintains developer productivity during testing

---

10. Administrative Command Reference

Basic Operational Commands

`/pause`

What It Is / How It Works:

- Sets `bot_active` flag to `false` in `system_state` table
- `SignalService.can_send_signal` checks flag before processing
- Rejected signals receive "Bot is currently paused by an admin" message

What It Does: Immediately halts all signal processing while maintaining system availability for status checks and configuration.

Administrator Value:

- Emergency stop capability during unexpected market conditions
- Maintenance window creation without system shutdown
- Prevents signal processing during configuration changes
- Instant activation (no service restart required)

`/resume`

What It Is / How It Works:

- Sets `bot_active` flag to `true` in `system_state` table
- Immediately enables signal processing

What It Does: Re-enables signal processing after administrative pause.

Administrator Value:

- Safe resumption after maintenance or configuration
- No accumulated signal backlog (signals during pause are rejected)
- Immediate effect without lag or startup time

`/stats`

What It Is / How It Works: Queries `signals` table for current UTC day and calculates:

```
- Total signals sent vs. MAX_SIGNALS_PER_DAY
- BUY signal count
- SELL signal count
- Closed trades count
- Win/Loss breakdown (positive vs. negative P&L)
- Total Profit & Loss for closed trades
- Current bot_active status
```

What It Does: Provides real-time performance dashboard for daily trading activity.

Administrator Value:

- Instant health check without database access
- Performance monitoring from mobile device
- Rapid identification of rate limit proximity
- Win rate and P&L tracking for strategy validation
- No external analytics platform required

Interactive Configuration Commands

`/set`

What It Is / How It Works:

Interactive conversation flow:

1. Display inline keyboard with changeable settings
2. Admin selects setting to modify
3. Bot displays current value and input format instructions
4. Admin submits new value
5. Bot validates input format and constraints
6. On validation success: Save to `settings` table, reload configuration in memory
7. On validation failure: Re-prompt with error message

Configurable Parameters:

- `MAX_SIGNALS_PER_DAY`: Integer >= 1

- `MIN_SECONDS_BETWEEN_SIGNALS`: Integer >= 0
- `TRADING_START_TIME`: HH:MM format (24-hour)
- `TRADING_END_TIME`: HH:MM format (24-hour)

What It Does: Enables zero-downtime modification of operational parameters through guided, validated input process.

Administrator Value:

- Adjust rate limits in response to market conditions without downtime
- Modify trading hours for different market sessions or strategies
- Inline keyboard prevents command syntax errors
- Input validation prevents configuration errors
- Immediate effect (no restart or reconnection required)
- Database persistence ensures settings survive system restarts

`/chats`

What It Is / How It Works:

Menu-driven conversation with three operations:

List Current Chats:

- Queries `managed_chats` table
- Calls Telegram API to refresh chat names
- Updates database with current names
- Displays formatted list with IDs

Add New Chat:

- Prompts for chat ID
- Validates ID format (numeric or @username format)
- Verifies bot membership in target chat
- Retrieves and stores chat name from Telegram API
- Adds to `managed_chats` table

Remove Chat:

- Displays inline keyboard of current chats
- Prevents removal of `TELEGRAM_DEFAULT_CHAT_ID`
- Removes selected chat from `managed_chats` table

What It Does: Manages the list of Telegram channels and groups receiving signal alerts through a safe, validated interface.

Administrator Value:

- Dynamic subscriber management without code changes
- Automatic chat name synchronization prevents stale labels
- Protection against accidental removal of primary channel
- Bot membership verification prevents configuration errors
- Enables multi-tier notification (subscriber channels, internal channels, test channels)
- No database access required for channel management

`/cancel`

What It Is / How It Works:

- Exits current conversation handler
- Resets conversation state
- Returns bot to idle command-listening mode

What It Does: Provides escape mechanism from interactive command sessions.

Administrator Value:

- Prevents locked conversation states
- Enables safe command sequence abort
- No timeout waiting required

System Information Commands

`/help`

What It Is / How It Works:

- Displays formatted list of all available commands
- Shows current rate limit parameters (`MAX_SIGNALS_PER_DAY`, `MIN_SECONDS_BETWEEN_SIGNALS`)
- Includes brief description of each command's purpose

What It Does: Provides quick reference for command syntax and current operational parameters.

Administrator Value:

- Eliminates need to reference external documentation
- Shows current rate limits for context
- Mobile-friendly command reference

---

11. Reports and System Diagnostics

What It Is / How It Works

Automatic Report Generation:

Reports created by `QueueService` under two conditions:

1. `RETRY_FAILURE` Reports:

- Signal fails processing initially (database lock, network error)
- Enters retry queue with `MAX_RETRIES` (5) attempts
- If all retries fail: Report logged with full signal details
- Admin notification sent immediately

2. `STALE_SIGNAL` Reports:

- Signal remains in retry queue longer than `SIGNAL_EXPIRY_MINUTES` (3)
- Discarded without further retries
- Report logged with signal details
- Admin notification sent

Report Structure:

```
id          # Unique identifier
timestamp   # Creation time
report_type # STALE_SIGNAL or RETRY_FAILURE
details     # JSON payload with signal data and error context
is_read     # Boolean acknowledgment flag
```

`/reports` Command Flow

What It Is / How It Works:

1. Query `reports` table for `is_read = FALSE`
2. If none: Display "No unread reports"
3. If present: Display inline keyboard with timestamp + type buttons
4. Admin selects report
5. Retrieve and display full `details` field
6. Set `is_read = TRUE` for selected report
7. Report removed from next `/reports` query

What It Does: Provides structured access to automatically logged system failures requiring admin attention.

Administrator Value

Solves: The challenge of identifying and diagnosing transient failures that don't warrant immediate system shutdown but require investigation.

Benefits:

- Complete failure context (signal data, error messages, timestamps)
- Automatic logging eliminates need for manual log parsing
- Read/unread tracking prevents duplicate investigation
- Notification system ensures awareness of critical failures
- Historical record of system issues for pattern identification
- Diagnostic payload includes exact signal data for reproduction
- Enables proactive issue resolution before impact accumulates
- Separates actionable failures from normal operation logs

System Deployment Notes

Prerequisites:

- Python 3.8+
- `.env` file with required credentials
- Valid Telegram Bot Token
- SSL certificates (for TCP server)

Startup Sequence:

1. Load `.env` configuration
2. Initialize SQLite database
3. Override settings from database `settings` table
4. Start FastAPI application
5. Launch TCP server in parallel task
6. Initialize Telegram bot
7. Start QueueService background worker

Security Considerations:

- `WEBHOOK_SECRET_KEY` must be cryptographically secure
- SSL certificates must be valid and properly secured
- `ADMIN_USER_IDS` should use Telegram user IDs (not usernames)
- Database file should have restricted filesystem permissions

---

End of Technical Administrator's Manual