QA-Testplan (Programmierer) — DowHowSignalService 2.010

Version: v2 (detailreicher Testkatalog) • Datum: 2026-01-10

Ziel: Den aktuellen Quellcode systematisch durchtesten (Ablauf, Objektmodell, Eventhandling, DB-Persistenz, Discord/Webhook, UI-Panels). Dieses Dokument ist für Entwickler gedacht: zu jedem Test steht *was* geprüft wird, *wo* im Code die Logik sitzt, *wie* man testet und *woran* man Pass/Fail erkennt.

Testumgebung (ausfüllen)	
MT5 Build / Broker	_____
EA-Datei	Trade Assistent V2.013.mq5/.ex5
Symbol / TF	_____
Inputs (relevante)	InpDebug=____ InpRequireKnownSymbol=____ Webhooks gesetzt? ____
Log-Datei	MQL5/Files/DowHowSignalService.log (Logger: CLogger::SetMethod(LOGGING_METHOD
DB-Datei	MQL5/Files/DowHowState_<SYMBOL>_<TF>.sqlite (CDBService::DefaultFile)

Wie man Logs/DB korrekt interpretiert (wichtig für alle Tests)

Der EA kann parallel auf mehreren Charts laufen. Damit du nicht aus Versehen die Logs/DB eines anderen Charts interpretierst, musst du bei jedem Test sicherstellen, dass Log-Datei, DB-Datei und (falls aktiv) Discord-Nachrichten zu **dem Chart** passen, den du gerade testest.

Konkret bedeutet das:

- **DB-Datei muss Symbol/TF enthalten:** Wenn du auf EURUSD H1 testest, muss die angelegte Datei **DowHowState_EURUSD_H1.sqlite** heißen (siehe **CDBService::DefaultFile()**).
- **Logs müssen den Kontext zeigen:** In der Log-Datei muss bei Debug-Ausgaben (z.B. **CDBService::PrintData()**) die Zeile **DB file: ...DowHowState_EURUSD_H1.sqlite** auftauchen. Das ist dein „Beweis“, dass du gerade die richtige Instanz anschaust.
- **Discord-Messages müssen Symbol/TF tragen:** Bei Line-Drag-Updates und Send-Signals steht Symbol und TF im Text (z.B. „EURUSD H1 Trade …“). Das muss zum Chart passen.
- **Wenn etwas nicht passt:** Du liest sehr wahrscheinlich Logs/DB einer anderen Chart-Instanz oder hast mehrere Instanzen laufen. Dann: erst alle EAs vom Chart entfernen, Logs leeren, neu starten.

Praktischer Ablauf zum „Logs passen zu meinem Test“-Check:

1. Nur **ein** Chart öffnen (für den Start). EA anhängen.
2. Im Terminal → **Experts**: nach „EA Startet OnInit()“ suchen.
3. InpDebug=true setzen → EA neu starten → in **DowHowSignalService.log** muss „DB file: ...DowHowState__.sqlite“ erscheinen.
4. Im Ordner **MQL5/Files** prüfen, ob genau diese SQLite-Datei existiert und Zeitstempel zur letzten Ausführung passt.

Testkatalog

Hinweis: Viele Tests sind so formuliert, dass sie auch dann funktionieren, wenn du gerade keine echten Trades eröffnest. Der Fokus liegt auf UI/Objekten, DB-Status und Discord-Notifications.

A) Start/Init/Deinit (Lebenszyklus)

INIT-01 — OnInit – erfolgreicher Startpfad

Was wird getestet?	Der komplette Startpfad läuft ohne INIT_FAILED durch: Logger initialisiert, DB init, Router/Discord init, Restore aus DB, UI-Panels/Buttons/Linien werden erzeugt.

Code-Referenzen	`Trade Assistent V2.013.mq5::OnInit()`
	`logger.mqh::CLogger::SetLogFileName/SetMethod/Add`
	`db_service.mqh::CDBService::Init()`
	`webhook_router.mqh::CWebhookRouter::Init/Add/Validate`
	`discord_client.mqh::CDiscordClient::Init()`
	`trade_manager.mqh::CTradeManager::Init()`
	`ui_manager.mqh::CUIManager::Init/RestoreTradeLinesFromDB`
	`createEntryAndSLLinien(), TA_Controllers_Init(), g_tp.Create/TP_RebuildRows()`

Voraussetzungen	• Nur 1 Chart geöffnet (für eindeutige Logs). • WebRequest für Discord-Domains freigegeben (falls Discord aktiv getestet wird). • Input-Webhooks gesetzt oder InpRequireKnownSymbol=false (je nach Test).
Schritte	1 EA an Chart anhängen. 2 Experts-Tab + DowHowSignalService.log prüfen. 3 Visuell prüfen: Entry/SL Buttons + HL-Linien vorhanden, Panel links (TP_) vorhanden.
Erwartet	• Im Log: „EA Startet OnInit()“ und danach keine INIT_FAILED-typischen Abbrüche. • DB-Datei im Ordner MQL5/Files vorhanden: DowHowState__.sqlite. • Keine Spam-Errors (ObjectCreate fails, DatabaseOpen fails, WebRequest denied).
Hinweise / Fehlerbilder	• Fail-Indikator: EA verschwindet direkt wieder vom Chart oder Journal zeigt INIT_FAILED. • Typisch bei Discord: „WebRequest is not allowed“ → dann Domain whitelisten. • Typisch bei DB: „DatabaseOpen failed“ → Pfad/Permissions prüfen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

INIT-02 — OnInit – DB Restore wird ausgeführt

Was wird getestet?	Beim Start wird der vorherige Zustand aus SQLite geladen (Meta + Positionen + Linienpreise).

Code-Referenzen	`Trade Assistent V2.013.mq5::OnInit() → g_TradeMgr.RestoreFromDB(_Symbol,_Period)`
	`db_service.mqh::CDBService::PrintData(), LoadPositions(), GetMetaInt/SetMetaInt`
	`ui_manager.mqh::CUIManager::RestoreTradeLinesFromDB()`

Voraussetzungen	• InpDebug=true (damit DB.PrintData() Log-Ausgaben schreibt). • Einmal vorher mindestens 1 Position/Line gespeichert (z.B. durch Drag/Send).
Schritte	1 EA starten, mindestens 1 Preisänderung per Drag ausführen (Entry oder SL oder TradePosLine). 2 EA vom Chart entfernen. 3 EA erneut anhängen. 4 DowHowSignalService.log prüfen (DB file / META / POS Einträge). 5 Visuell: Linien/Tags sollten wieder erscheinen und am gespeicherten Preis liegen.

Erwartet	• Log zeigt: „DB file: ...DowHowState__.sqlite“. • Log zeigt META- und POSITIONS-Zeilen (Anzahl >= 1, je nach vorherigem Zustand). • Nach Restart: TradePosLines/Tags werden wieder hergestellt (kein „Reset auf Default“).
Hinweise / Fehlerbilder	• Wenn META/POSITIONS immer 0 sind: Prüfen ob überhaupt gespeichert wird (Drag-Finalize ruft SaveLinePrices). • Wenn falsches Symbol/TF: du hast mehrere DB-Dateien; prüfe Dateinamen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

DEINIT-01 — OnDeinit – vollständiges Cleanup

Was wird getestet?	Beim Entfernen des EA werden registrierte UI-Objekte gelöscht, Panels zerstört, Registry zurückgesetzt und DB geschlossen.

Code-Referenzen	`Trade Assistent V2.013.mq5::OnDeinit()`
	`ui_registry.mqh::UI_Reg_DeleteAll()`
	`trade_pos_line_registry.mqh::TradePosLines_Reset()`
	`trades_panel_class.mqh::g_tp.Destroy()`
	`db_service.mqh::CDBService::Close()`

Voraussetzungen	• EA läuft und hat UI-Objekte erstellt (INIT-01 bestanden).
Schritte	1 EA vom Chart entfernen. 2 Im Log prüfen: Zeile „OnDeinit(reason=...): UI Objekte gelöscht=...“ 3 Chart visuell prüfen: Entry/SL/TP_-Objekte sollten verschwunden sein (keine Reste).
Erwartet	• UI_Reg_DeleteAll() entfernt die registrierten Objekte (deleted > 0 in typischen Runs). • Keine „dangling tags“ (Tags ohne Linien) verbleiben im Chart. • DB wird ohne Fehler geschlossen (keine DatabaseFinalize-Leaks sichtbar).
Hinweise / Fehlerbilder	• Wenn Objekte bleiben: sind sie evtl. nicht im Registry registriert → Registry/Creation prüfen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

B) Log/DB/Discord-Konsistenz (dein "Logs spiegeln Inputs"-Punkt)

CONS-01 — DB-Dateiname passt zu Chart-Symbol/TF

Was wird getestet?	Sicherstellen, dass du beim Testen nicht aus Versehen eine andere EA-Instanz/DB ausliest. Der Dateiname ist der einfachste harte Beweis.
Code-Referenzen	`db_service.mqh::CDBService::DefaultFile()` `db_service.mqh::CDBService::Init() / PrintData() (loggt „DB file: …“)`
Voraussetzungen	• InpDebug=true (für PrintData-Log).
Schritte	1 Chart auf ein eindeutiges Symbol/TF stellen (z.B. EURUSD H1). 2 EA starten (OnInit). 3 Im Ordner MQL5/Files prüfen: existiert DowHowState_EURUSD_H1.sqlite? 4 In DowHowSignalService.log nach „DB file:“ suchen.
Erwartet	• DB-Dateiname enthält genau das Symbol und TF des Charts. • Log-Zeile „DB file:“ zeigt denselben Dateinamen.
Hinweise / Fehlerbilder	• Wenn Datei anders heißt: du testest nicht das, was du glaubst (falsches Chart, mehrere Instanzen, falsches Symbol alias).

Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

CONS-02 — Discord-Update nach Linien-Drag trägt richtigen Kontext

Was wird getestet?	Wenn du eine TradePosLine (Entry/SL einer Position) verschiebst, muss die Discord-Nachricht Symbol/TF/Trade/Pos enthalten, die zu deiner Testaktion passen.

Code-Referenzen	`CTradePosLineDragController.mqh::Finalize()`
	`trade_manager.mqh::CTradeManager::SaveLinePrices()`
	`discord_client.mqh::CDiscordClient::SendMessage()`

Voraussetzungen	• Discord-Webhook aktiv (g_Discord.Init erfolgreich). • Mindestens 1 TradePosLine existiert (z.B. LONG Trade 1 Pos 1 Entry/SL Linie).
Schritte	1 Eine TradePosLine (z.B. LONG Trade 1 Pos 1 Entry) anklicken und deutlich verschieben. 2 Maus loslassen (Finalize muss feuern). 3 Discord-Channel prüfen (UPDATE Nachricht).
Erwartet	• Message enthält: Symbol (z.B. EURUSD) und TF (z.B. H1) und Trade/Pos Nummern. • Preisänderung wird korrekt angezeigt (old -> new). • Nachricht kommt nur, wenn Preis sich wirklich geändert hat (Toleranz pt*0.25).
Hinweise / Fehlerbilder	• Wenn jede kleinste Bewegung spammt: changed-Logik/pt Toleranz prüfen. • Wenn keine Nachricht: g_Discord.Init fehlgeschlagen oder WebRequest nicht erlaubt.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

C) TradePosLines Objektmodell (CTradePosLine + Registry + DragController)

TPL-01 — CTradePosLine::Create erzeugt Linie + Tag (lesbar im Chart)

Was wird getestet?	Eine TradePosLine besteht aus HL-Linie + Tag-Label. Tag muss sichtbar im Chart liegen (nicht rechts außerhalb).
Code-Referenzen	`CTradePosLine.mqh::Create(), CreateTagIfNeeded(), SyncTagToLine()` `CTradePosLine.mqh::SetTagShiftBars(), SetTagOffsetPx(), SetTagCorner()`
Voraussetzungen	• Chart mit genügend Historie (nicht ganz nach rechts gescrollt). • Mindestens 1 TradePosLine wird erzeugt (durch Restore oder manuell über Codepfad).
Schritte	1 TradePosLine erzeugen lassen (z.B. durch RestoreTradeLinesFromDB oder durch Send/Panel-Funktion). 2 Visuell prüfen: Linie vorhanden; Tag-Text vorhanden und lesbar. 3 Chart zoomen (rein/raus) und etwas scrollen.
Erwartet	• Tag bleibt nahe der Linie und bleibt innerhalb der sichtbaren Chartfläche. • Tag klebt an der Linie (SyncTagToLine nutzt ChartTimePriceToXY).
Hinweise / Fehlerbilder	• Wenn Tag weit rechts „klebt“: TagShiftBars/Offset zu groß oder corner falsch. • Wenn Tag „verschwindet“: ChartTimePriceToXY liefert false (Zeit außerhalb sichtbarer Range).
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

TPL-02 — Linie startet gestrichelt, kann auf durchgezogen wechseln

Was wird getestet?	Neue TradePosLines sind initial gestrichelt (Style=DOT). Später kann auf durchgezogen gewechselt werden (z.B. bei Statuswechsel PENDING→OPEN).
Code-Referenzen	`CTradePosLine.mqh::SetDashed(), SetSolid()` `sonstige_methoden.mqh::TPSLReached() → g_TradeMgr.SetPosLinesSolid(...) (bei Entry Hit)`
Voraussetzungen	• Mindestens 1 TradePosLine existiert. • Optional: Statuswechsel durch Testdaten oder manuell in DB erzwingen.
Schritte	1 Nach Erstellung prüfen: Linie ist gestrichelt. 2 Statuswechsel simulieren: entweder Entry-Hit abwarten oder (für QA) in DB status='OPEN' setzen und Restore laufen lassen. 3 Prüfen: Linie wird durchgezogen dargestellt.
Erwartet	• DOT-Style bei neuen Linien, SOLID-Style nach SetSolid. • Style-Wechsel betrifft die richtige Linie (nicht PR_HL/SL_HL der Basis-Buttons).
Hinweise / Fehlerbilder	• Wenn falsche Linie umgestellt wird: Namensparsing (UI_IsTradePosLine / Registry Lookup) prüfen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

TPL-03 — Drag – Tag bewegt sich ohne spürbare Verzögerung

Was wird getestet?	Beim Drag einer TradePosLine muss das Tag nahezu synchron nachlaufen (keine Sekunden-Verzögerung).
Code-Referenzen	`CTradePosLineDragController.mqh::OnChartEvent() / OnMouseMove() / ThrottledTagSync()` `CTradePosLine.mqh::SyncTagToLine()`

Voraussetzungen	• Eine TradePosLine mit Tag ist vorhanden.
Schritte	1 Linie anklicken und langsam ziehen. 2 Dabei auf das Tag achten: folgt es direkt oder „springt“ es hinterher? 3 Linie loslassen.
Erwartet	• Während Drag: Tag folgt (ThrottledTagSync: max ~30 FPS). • Nach Loslassen: Tag sitzt exakt am finalen Preis (Finalize ruft UI_CreateOrUpdateLineTag + Redraw).
Hinweise / Fehlerbilder	• Wenn es laggt: Throttle-Intervall zu hoch oder ChartRedraw wird zu selten ausgelöst. • Wenn es nach Loslassen falsch sitzt: Finalize/SyncTagToLine Fehler oder falscher Name in m_name.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

TPL-04 — Finalize – DB Save + Discord Update nur bei echter Änderung

Was wird getestet?	Nach Drag-Ende: (1) Tag final updaten, (2) Preise in DB speichern, (3) Discord senden nur wenn Preisänderung > pt*0.25.
Code-Referenzen	`CTradePosLineDragController.mqh::Finalize()` `trade_manager.mqh::CTradeManager::SaveLinePrices()`
Voraussetzungen	• Discord aktiv oder alternativ nur DB prüfen. • InpDebug=true um DB.PrintData zu nutzen.
Schritte	1 Eine TradePosLine minimal bewegen (< 0.25 Punkt) und loslassen. 2 Dann deutlich bewegen (> 1 Punkt) und loslassen. 3 InpDebug=true: DB.PrintData ausführen (EA restart) oder SQLite prüfen.
Erwartet	• Bei Mini-Bewegung: keine Discord-UPDATE Nachricht (changed=false). • Bei deutlicher Bewegung: Discord-UPDATE Nachricht erscheint. • DB enthält den finalen Preis (LoadPositions/PrintData zeigt neuen entry/sl).
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

D) Basis-UI (Entry/SL Gruppe + PR_HL/SL_HL)

BASE-01 — createEntryAndSLLinien – Basisobjekte werden erzeugt und sind bedienbar

Was wird getestet?	Entry/SL Buttons, Eingabefelder und PR_HL/SL_HL Linien existieren nach Start. Linien sind anklick- und dragbar.
Code-Referenzen	`Trade Assistent V2.013.mq5::OnInit()` → `createEntryAndSLLinien()` `event.mqh::OnChartEvent(...) (OBJ_DRAG/CLICK) – Basis-Linien-Handling`
Voraussetzungen	-
Schritte	1 EA starten. 2 PR_HL anklicken und verschieben. 3 SL_HL anklicken und verschieben.
Erwartet	• Beide HL-Linien lassen sich ziehen (OBJPROP_SELECTABLE/OBJPROP_SELECTED korrekt). • Die zugehörigen UI-Elemente (Button-Text/Preis) werden aktualisiert.

Hinweise / Fehlerbilder	• Wenn SL_HL nicht anklickbar: prüfen ob OBJPROP_SELECTABLE/OBJPROP_HIDDEN gesetzt ist.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

BASE-02 — Drag Basislinien synchronisiert Button-Text und interne Preise

Was wird getestet?	Beim Verschieben von PR_HL/SL_HL werden Entry_Price/SL_Price aktualisiert und die UI-Texte (Preis/Lot/Buy-Sell) werden konsistent angepasst.

Code-Referenzen	`event.mqh::MouseMove-States / UpdateEntrySLText (je nach Implementierung)`
	`trade_manager.mqh::calcLots() (falls Lot berechnet wird)`
	`Trade Assistent V2.013.mq5: Variablen Entry_Price, SL_Price`

Voraussetzungen	• Lot-/Text-Update muss aktiv sein (kein Debug-Stub).
Schritte	1 PR_HL nach oben/unten ziehen. 2 SL_HL so verschieben, dass SL > Entry wird und wieder zurück. 3 Buttontexte prüfen (Richtungswechsel Buy/Sell, Lot, Distanz).
Erwartet	• Preis im Text entspricht dem Linienpreis (ObjectGetDouble OBJPROP_PRICE). • Bei SL > Entry: Richtungslogik dreht wie vorgesehen. • Keine NaN/0 Werte, keine falschen Dezimalstellen.
Hinweise / Fehlerbilder	• Wenn Texte nicht stimmen: Reihenfolge der Updates (erst Preis lesen, dann Text setzen) prüfen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

E) Trades Panel (TP_*)

TP-01 — Panel Create + RebuildRows erzeugt konsistente UI

Was wird getestet?	Das neue Panel (g_tp) wird erstellt und die Rows werden anhand DB/Cache sauber aufgebaut.

Code-Referenzen	`Trade Assistent V2.013.mq5::OnInit() → g_tp.Create(), TP_RebuildRows(), g_tp.RebuildRows()`
	`trades_panel_class.mqh::Create(), RebuildRows(), Destroy()`
	`trades_panel.mqh::TP_DeleteByPrefix(), TP_RebuildRows() (falls Wrapper genutzt)`

Voraussetzungen	-
Schritte	1 EA starten. 2 Visuell: Panel links sichtbar, keine überlappenden Rows. 3 Mehrere Positionen erzeugen (SendDraft) und danach RebuildRows auslösen (z.B. durch Restart).
Erwartet	• Rows entsprechen DB-Positions: pro Position eine Zeile mit Buttons (Cancel/Stop). • Bei 0 Positionen: Panel zeigt leeren Zustand ohne Fehlermeldungen.
Hinweise / Fehlerbilder	• Wenn Rows doppelt erscheinen: Cleanup vor Create/Rebuild (TP_DeleteByPrefix) prüfen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

TP-02 — Row-Buttons: Cancel/Stop lösen TradeManager-Aktionen aus

Was wird getestet?	Klick auf Cancel/Stop in einer Row führt zu HandlePositionAction / CancelTrade und aktualisiert UI + DB + ggf. Discord.

Code-Referenzen	`trade_manager.mqh::HandlePositionAction(), CancelTrade()`
	`event.mqh::OnChartEvent – Button click routing`
	`db_service.mqh::UpsertPosition() (status update), LoadPositions()`

Voraussetzungen	• Mindestens 1 Position existiert (was_sent=1 oder testweise). • Discord optional aktiv.
Schritte	1 Im Panel eine Row auswählen. 2 Cancel klicken → prüfen was passiert (Status / Linien löschen). 3 Stop klicken → prüfen was passiert (Status / Linien löschen). 4 EA restart → DB.PrintData prüfen (status verändert?).
Erwartet	• Aktion ändert Status in DB (z.B. CLOSED oder CLOSED_SL je nach Action). • TradePosLines werden gelöscht (DeletePosLines). • Panel aktualisiert sich (Row verschwindet oder Status ändert sich).
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

F) Business Rules (max 4 simultane Positionen, Re-Use nach Close)

RULE-01 — Max 4 gleichzeitig pro Richtung/Trade: 5. wird verhindert

Was wird getestet?	Es dürfen maximal 4 gleichzeitige aktive Positionen existieren. Die 5. aktive Erzeugung muss blockieren.
Code-Referenzen	`db_service.mqh::CountActivePositions(), GetNextPosNo(), LoadPositions()` `trade_manager.mqh::SendSignalDraft() / CreateDraft() (PosNo Vergabe)`
Voraussetzungen	• Panel/Send-Funktion erzeugt Positionen in DB.

Schritte	1 4 Positionen nacheinander erzeugen (z.B. LONG Trade 1 Pos1..Pos4). 2 Versuchen, eine 5. Position zu erzeugen. 3 DB.PrintData prüfen.
Erwartet	• Position 5 wird nicht als aktiv angelegt (kein zusätzlicher POS-Eintrag als aktiv). • UI zeigt keine 5. Row für aktive Positionen. • Ggf. Fehlermeldung/Return-Code (SEND_ERR_MAXPOS).
Hinweise / Fehlerbilder	• Wenn doch 5 existiert: CountActivePositions Filter auf Status/was_sent/is_pending prüfen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

RULE-02 — Nach Schließen darf wieder eine neue Position entstehen (trotz früherem Pos5)

Was wird getestet?	Wenn eine Position geschlossen wurde, darf wieder eine neue Position eröffnet werden, solange nicht >4 gleichzeitig aktiv sind. PosNo kann dabei >4 sein (historisch).
Code-Referenzen	`db_service.mqh::CountActivePositions() (nur aktive zählen)` `db_service.mqh::GetNextPosNo() (MAX(pos_no)+1 historisch)`
Voraussetzungen	• Mindestens 4 Positionen existieren, mindestens 1 davon geschlossen.
Schritte	1 4 Positionen anlegen. 2 Eine Position schließen (Cancel/Stop oder status auf CLOSED setzen). 3 Neue Position anlegen. 4 DB.PrintData prüfen: Anzahl aktiver Positionen <=4.
Erwartet	• Neue Position wird erlaubt (weil aktive < 4). • PosNo kann 5 sein (historisch), aber gleichzeitig aktiv bleiben max. 4.
Hinweise / Fehlerbilder	• Das ist kein Bug: PosNo ist eine laufende Nummer, nicht „Slot 1..4“. Relevant ist nur die Anzahl aktiver Positionen.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

G) Negativ-/Fehlerpfad-Tests

ERR-01 — Discord WebRequest nicht erlaubt → sauberer Fehler

Was wird getestet?	Wenn WebRequest nicht freigeschaltet ist, muss der EA den Fehler klar loggen und (je nach Design) Init abbrechen oder Discord deaktivieren.
Code-Referenzen	`discord_client.mqh::Init(), TestConnection(), SendMessage()` `MT5 WebRequest Permissions`
Voraussetzungen	• In MT5: WebRequest-Domains NICHT freigeben.
Schritte	1 EA starten (Discord aktiv). 2 Journal/Experts prüfen.
Erwartet	• Klarer Hinweis in Logs (WebRequest denied / error code 4014). • Kein stilles „Nichts passiert“ ohne Log.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

ERR-02 — Unbekanntes Symbol bei InpRequireKnownSymbol=true → Init blockiert

Was wird getestet?	Router-Validierung muss verhindern, dass ein unbekanntes Symbol ohne Webhook-Mapping läuft (um falsche Discord-Ziele zu vermeiden).
Code-Referenzen	`webhook_router.mqh::CWebhookRouter::Validate()` `Trade Assistent V2.013.mq5::OnInit() → g_router.Validate()`
Voraussetzungen	• InpRequireKnownSymbol=true • Chart auf ein Symbol ohne Mapping stellen (z.B. exotisches Symbol).
Schritte	1 EA starten. 2 Journal/Experts prüfen, ob INIT_FAILED kommt.
Erwartet	• Init bricht mit klarer Log-Meldung ab (fehlendes Mapping). • Kein späteres Senden an falsche Webhooks.
Ergebnis	[ ] PASS [ ] FAIL Notizen: _____

Abschluss / Sign-off

Wenn alle Tests PASS sind, ist der aktuelle Stand regressions-sicher für: Start/Restore/Cleanup, TradePosLines-Objektmodell, Drag-Sync, DB-Persistenz und Discord-Updates. Empfehlung: Logs/DB-Artefakte (Log-Datei + SQLite) zusammen mit Screenshots pro Fail-Fall sammeln.

Sign-off

Tester: _____ Datum: _____ Build: _____