```
/***** TradingOps One-Click v4 (Enhanced) - America/New_York *****/
const TZ = "America/New_York";
const SH = {
  CFG: 'Config',
  WL: 'Watchlist',
  TICKS: 'Ticks',
  AL: 'Alerts',
  HL: 'Health',
  DASH: 'Dashboard'
};

// Cache for configuration values to reduce sheet reads
const CONFIG_CACHE = {};
const CACHE_DURATION_MS = 60000; // 1 minute

/* ---------- Utils ---------- */
function ss_() {
  return SpreadsheetApp.getActive();
}

function sh_(n) {
  return ss_().getSheetByName(n) || ss_().insertSheet(n);
}

function now_() {
  return Utilities.formatDate(new Date(), TZ, "yyyy-MM-dd HH:mm:ss");
}

/**
 * Get configuration value with caching
```

```
 * @param {string} k - Configuration key
 * @param {boolean} forceRefresh - Force cache refresh
 * @return {string} Configuration value
 */
function cfg_(k, forceRefresh = false) {
  const cacheKey = `cfg_${k}`;
  const cached = CONFIG_CACHE[cacheKey];

  if (!forceRefresh && cached && (Date.now() - cached.timestamp < CACHE_DURATION_MS)) {
    return cached.value;
  }

  try {
    const s = sh_(SH.CFG);
    const vals = s.getDataRange().getValues().filter(r => r[0]);
    const m = Object.fromEntries(vals.map(r => [String(r[0]), String(r[1] || '')]));
    const value = m[k] || '';

    CONFIG_CACHE[cacheKey] = { value: value, timestamp: Date.now() };
    return value;
  } catch (err) {
    logError_('cfg_', err, `Failed to get config for key: ${k}`);
    return '';
  }
}

/**
 * Set health metric value
 * @param {string} key - Metric name
 * @param {*} val - Metric value
 */
```

```
function setHL_(key, val) {
  try {
    const s = sh_(SH.HL);
    const last = s.getLastRow();

    if (last === 0) {
      s.appendRow(['Metric', 'Value', 'Updated']);
    }

    let r = -1;
    const data = s.getDataRange().getValues();
    for (let i = 1; i < data.length; i++) {
      if (data[i][0] === key) {
        r = i + 1;
        break;
      }
    }

    const timestamp = now_();
    if (r === -1) {
      s.appendRow([key, val, timestamp]);
    } else {
      s.getRange(r, 2, 1, 2).setValues([[val, timestamp]]);
    }
  } catch (err) {
    logError_('setHL_', err, `Failed to set health metric: ${key}`);
  }
}

/**
 * Log errors to Alerts sheet
```

```
 * @param {string} func - Function name
 * @param {Error} err - Error object
 * @param {string} context - Additional context
 */
function logError_(func, err, context = '') {
  try {
    const al = sh_(SH.AL);
    const msg = `ERROR in ${func}: ${err.message || err}${context ? ' | ' + context : ''}`;
    al.appendRow([now_(), 'ERROR', func, '', msg]);
    console.error(msg);
  } catch (e) {
    console.error('Failed to log error:', e);
  }
}

/* ---------- Bootstrap (tabs, headers, defaults) ---------- */
function Bootstrap_() {
  try {
    // Config
    const cfg = sh_(SH.CFG);
    cfg.clear();
    cfg.appendRow(['Key', 'Value', 'Description']);
    cfg.appendRow(['POLL_INTERVAL_MIN', '1', 'Minutes between price updates (1, 5, 10, 15, or 30)']);
    cfg.appendRow(['WEBHOOK_SECRET', '', 'Optional secret for webhook authentication']);
    cfg.appendRow(['MAX_TICK_ROWS', '10000', 'Maximum rows to keep in Ticks sheet (auto-prune)']);
    cfg.appendRow(['ALERT_ON_ERROR', 'TRUE', 'Send email on critical errors (TRUE/FALSE)']);
    cfg.appendRow(['RETRY_ATTEMPTS', '3', 'Number of retry attempts for API calls']);
    cfg.appendRow(['FMP_API_KEY', '', 'OPTIONAL - Financial Modeling Prep API key (Yahoo Finance used by default)']);

    // Format config sheet
    cfg.getRange(1, 1, 1, 3).setFontWeight('bold').setBackground('#4285f4').setFontColor('white');
```

```
cfg.setColumnWidths(1, 1, 200);
cfg.setColumnWidths(2, 1, 300);
cfg.setColumnWidths(3, 1, 400);

// Watchlist
const wl = sh_(SH.WL);
wl.clear();
wl.appendRow(['Symbol', 'AssetClass', 'Venue', 'Notes', 'Active']);
wl.appendRow(['SPY', 'STOCK', 'YAHOO', 'S&P 500 ETF', 'TRUE']);
wl.appendRow(['QQQ', 'STOCK', 'YAHOO', 'Nasdaq 100 ETF', 'TRUE']);
wl.appendRow(['NVDA', 'STOCK', 'YAHOO', 'NVIDIA Corp', 'TRUE']);
wl.appendRow(['AAPL', 'STOCK', 'YAHOO', 'Apple Inc', 'TRUE']);
wl.appendRow(['BTCUSDT', 'CRYPTO', 'BINANCE', 'Bitcoin', 'TRUE']);
wl.appendRow(['ETHUSDT', 'CRYPTO', 'BINANCE', 'Ethereum', 'TRUE']);
wl.appendRow(['SOLUSDT', 'CRYPTO', 'BINANCE', 'Solana', 'TRUE']);

// Format watchlist
wl.getRange(1, 1, 1, 5).setFontWeight('bold').setBackground('#34a853').setFontColor('white');
wl.setColumnWidths(1, 1, 120);
wl.setColumnWidths(2, 1, 100);
wl.setColumnWidths(3, 1, 100);
wl.setColumnWidths(4, 1, 200);
wl.setColumnWidths(5, 1, 80);

// Ticks
const t = sh_(SH.TICKS);
t.clear();
t.appendRow(['Timestamp_ET']);
t.getRange(1, 1, 1, 1).setFontWeight('bold').setBackground('#fbbc04').setFontColor('black');

// Alerts
```

```
    const al = sh_(SH.AL);
    al.clear();
    al.appendRow(['Timestamp_ET', 'Source', 'Symbol', 'Price', 'Payload']);
    al.getRange(1, 1, 1, 5).setFontWeight('bold').setBackground('#ea4335').setFontColor('white');
    al.setFrozenRows(1);

    // Health
    const hl = sh_(SH.HL);
    hl.clear();
    hl.appendRow(['Metric', 'Value', 'Updated']);
    hl.getRange(1, 1, 1, 3).setFontWeight('bold').setBackground('#4285f4').setFontColor('white');

    setHL_('Last Price Poll (ET)', 'Never');
    setHL_('Last Alert (ET)', 'Never');
    setHL_('Total Ticks Collected', '0');
    setHL_('Active Symbols', '0');
    setHL_('API Errors (24h)', '0');
    setHL_('Web App URL', 'Deploy via Extensions → Apps Script');
    setHL_('Data Sources', 'Yahoo Finance (stocks) + Binance (crypto) - 100% FREE');
    setHL_('Status', 'Initialized - Run QuickStart');

    BuildDashboard_();

    SpreadsheetApp.getActiveSpreadsheet().toast('Bootstrap complete!', 'Success', 3);
  } catch (err) {
    logError_('Bootstrap_', err);
    SpreadsheetApp.getUi().alert('Bootstrap failed: ' + err.message);
  }
}

/* ---------- Ensure Tick columns match Watchlist ---------- */
```

```
function EnsureTickColumns_() {
  try {
    const wl = sh_(SH.WL).getDataRange().getValues().slice(1).filter(r => r[0] && r[4] !== 'FALSE');
    const syms = wl.map(r => String(r[0]).toUpperCase());
    const t = sh_(SH.TICKS);

    if (t.getLastRow() === 0) {
      t.appendRow(['Timestamp_ET']);
    }

    const lastCol = Math.max(1, t.getLastColumn());
    const header = t.getRange(1, 1, 1, lastCol).getValues()[0];
    const have = new Set(header.slice(1));

    syms.forEach(sym => {
      if (!have.has(sym)) {
        const newCol = t.getLastColumn() + 1;
        t.getRange(1, newCol).setValue(sym);
        t.getRange(1, newCol).setFontWeight('bold').setBackground('#fbbc04');
      }
    });

    return syms;
  } catch (err) {
    logError_('EnsureTickColumns_', err);
    return [];
  }
}

/* ---------- Fetchers ---------- */
```

```
/**
 * Fetch Binance crypto prices using parallel requests
 * @param {Array<string>} syms - Array of symbols
 * @return {Object} Map of symbol to price
 */
function fetchBinance_(syms) {
  if (!syms.length) return {};

  try {
    // Build all requests upfront
    const requests = syms.map(sym => ({
      url: `https://api.binance.com/api/v3/ticker/price?symbol=${encodeURIComponent(sym)}`,
      muteHttpExceptions: true
    }));

    // Fetch all in parallel - MUCH faster!
    const responses = UrlFetchApp.fetchAll(requests);

    const priceMap = {};
    responses.forEach((response, index) => {
      const sym = syms[index];
      try {
        if (response.getResponseCode() === 200) {
          const data = JSON.parse(response.getContentText());
          priceMap[sym] = data?.price ? Number(data.price) : null;
        } else {
          priceMap[sym] = null;
        }
      } catch (err) {
        console.warn(`Failed to parse ${sym}:`, err.message);
        priceMap[sym] = null;
```

```
      }
    });

    return priceMap;
  } catch (err) {
    logError_('fetchBinance_', err, `Symbols: ${syms.join(',')}`);
    return Object.fromEntries(syms.map(s => [s, null]));
  }
}

/**
 * Fetch stock prices from Yahoo Finance (FREE - no API key needed)
 * Uses UrlFetchApp.fetchAll for parallel requests
 * @param {Array<string>} syms - Array of symbols
 * @return {Object} Map of symbol to price
 */
function fetchYahoo_(syms) {
  if (!syms.length) return {};

  try {
    // Build all requests upfront
    const requests = syms.map(sym => ({
      url: `https://query1.finance.yahoo.com/v8/finance/chart/${encodeURIComponent(sym)}?interval=1d&range=1d`,
      muteHttpExceptions: true
    }));

    // Fetch all in parallel - MUCH faster!
    const responses = UrlFetchApp.fetchAll(requests);

    const map = {};
    responses.forEach((response, index) => {
```

```
      const sym = syms[index];
      try {
        if (response.getResponseCode() === 200) {
          const json = JSON.parse(response.getContentText());
          const result = json?.chart?.result?.[0];
          const price = result?.meta?.regularMarketPrice;

          map[sym] = (price != null) ? Number(price) : null;
        } else {
          map[sym] = null;
        }
      } catch (err) {
        console.warn(`Failed to parse ${sym}:`, err.message);
        map[sym] = null;
      }
    });

    return map;
  } catch (err) {
    logError_('fetchYahoo_', err, `Symbols: ${syms.join(',')}`);
    return Object.fromEntries(syms.map(s => [s, null]));
  }
}

/* ---------- Poller: writes one wide row in TICKS ---------- */
function UpdateTicks_() {
  try {
    const wl = sh_(SH.WL).getDataRange().getValues().slice(1).filter(r => r[0] && r[4] !== 'FALSE');

    if (!wl.length) {
      setHL_('Status', 'No active symbols in watchlist');
```

```
  return;
}

const syms = EnsureTickColumns_();
const t = sh_(SH.TICKS);
const header = t.getRange(1, 1, 1, t.getLastColumn()).getValues()[0];

const stock = wl.filter(r => r[1] === 'STOCK').map(r => String(r[0]).toUpperCase());
const crypto = wl.filter(r => r[1] === 'CRYPTO').map(r => String(r[0]).toUpperCase());

// Fetch prices - Yahoo Finance for stocks (FREE), Binance for crypto (FREE)
const mapS = fetchYahoo_(stock);
const mapC = fetchBinance_(crypto);

// Build row
const row = [now_()];
header.slice(1).forEach(sym => {
  row.push((sym in mapS) ? mapS[sym] : ((sym in mapC) ? mapC[sym] : null));
});

// Append row
t.appendRow(row);

// Auto-prune old data
const maxRows = parseInt(cfg_('MAX_TICK_ROWS') || '10000', 10);
const currentRows = t.getLastRow();
if (currentRows > maxRows + 1) {
  const deleteCount = currentRows - maxRows - 1;
  t.deleteRows(2, deleteCount); // Keep header
}
```

```
    // Update health metrics
    setHL_('Last Price Poll (ET)', now_());
    setHL_('Total Ticks Collected', String(t.getLastRow() - 1));
    setHL_('Active Symbols', String(syms.length));
    setHL_('Status', 'Running');

    console.log(`Updated ${syms.length} symbols at ${now_()}`);
  } catch (err) {
    logError_('UpdateTicks_', err);
    setHL_('Status', 'Error - Check Alerts sheet');

    if (cfg_('ALERT_ON_ERROR') === 'TRUE') {
      sendErrorEmail_(err, 'UpdateTicks_');
    }
  }
}

/**
 * Send error notification email
 * @param {Error} err - Error object
 * @param {string} func - Function name
 */
function sendErrorEmail_(err, func) {
  try {
    const email = Session.getActiveUser().getEmail();
    if (email) {
      MailApp.sendEmail({
        to: email,
        subject: `TradingOps Error Alert: ${func}`,
        body: `An error occurred in ${func} at ${now_()}\n\nError: ${err.message}\n\nStack: ${err.stack}`
      });
```

```
    }
  } catch (e) {
    console.error('Failed to send error email:', e);
  }
}

/* ---------- Webhook (TradingView & others) ---------- */
function doPost(e) {
  try {
    // Validate secret if configured
    const sec = cfg_('WEBHOOK_SECRET');
    const got = (e?.headers?.['X-Webhook-Secret']) ||
                (e?.headers?.['x-webhook-secret']) ||
                (e?.parameter?.secret) || '';

    if (sec && String(sec) !== String(got)) {
      sh_(SH.AL).appendRow([now_(), 'DENY', '', '', 'Invalid webhook secret']);
      return ContentService.createTextOutput('forbidden').setMimeType(ContentService.MimeType.TEXT);
    }

    // Parse payload
    const raw = e?.postData?.contents || '{}';
    let o = {};
    try {
      o = JSON.parse(raw);
    } catch (parseErr) {
      o = { payload: raw };
    }

    const sym = String(o.symbol || o.ticker || '').toUpperCase();
    const px = Number(o.price || o.close || o.last || NaN);
```

```
  const action = String(o.action || o.signal || '');

  // Enhanced logging
  sh_(SH.AL).appendRow([
    now_(),
    'POST',
    sym,
    isFinite(px) ? px : '',
    JSON.stringify({
      action: action,
      price: px,
      raw: raw.substring(0, 500) // Limit payload size
    })
  ]);

  setHL_('Last Alert (ET)', now_());

  return ContentService.createTextOutput(JSON.stringify({
    status: 'success',
    timestamp: now_(),
    symbol: sym
  })).setMimeType(ContentService.MimeType.JSON);

} catch (err) {
  logError_('doPost', err);
  sh_(SH.AL).appendRow([now_(), 'ERROR', '', '', String(err)]);
  return ContentService.createTextOutput(JSON.stringify({
    status: 'error',
    message: err.message
  })).setMimeType(ContentService.MimeType.JSON);
}
```

```
}

function doGet(e) {
  try {
    const sym = (e?.parameter?.symbol || '').toUpperCase();
    const price = e?.parameter?.price || '';
    const action = e?.parameter?.action || '';

    sh_(SH.AL).appendRow([
      now_(),
      'GET',
      sym,
      price,
      JSON.stringify({ action: action, params: e?.parameter || {} })
    ]);

    setHL_('Last Alert (ET)', now_());

    return ContentService.createTextOutput(JSON.stringify({
      status: 'success',
      timestamp: now_()
    })).setMimeType(ContentService.MimeType.JSON);

  } catch (err) {
    logError_('doGet', err);
    return ContentService.createTextOutput(JSON.stringify({
      status: 'error',
      message: err.message
    })).setMimeType(ContentService.MimeType.JSON);
  }
}
```

```
/* ---------- Dashboard (charts auto) ---------- */
function BuildDashboard_() {
  try {
    const d = sh_(SH.DASH);
    d.clear();

    // Title and controls
    d.getRange('A1').setValue('TradingOps — Dashboard').setFontSize(18).setFontWeight('bold');
    d.getRange('A2').setValue('Rows to display:');
    d.getRange('B2').setValue(300).setFontWeight('bold');
    d.getRange('A3').setValue('Total tick rows:');
    d.getRange('B3').setFormula('=COUNTA(Ticks!A:A)-1').setFontWeight('bold');
    d.getRange('A4').setValue('Last update:');
    d.getRange('B4').setFormula('=IF(Ticks!A2="","Never",Ticks!A2)').setFontWeight('bold');

    // Data section
    d.getRange('A6').setValue('Recent Price Data (auto-refreshing)').setFontWeight('bold').setFontSize(12);
    d.getRange('A7').setFormula('=QUERY(Ticks!A:ZZ,"select * order by A desc limit "&$B$2,1)');

    // Format
    d.setColumnWidth(1, 200);
    d.setColumnWidth(2, 120);

    // Build charts if we have data
    const t = sh_(SH.TICKS);
    const cols = Math.max(2, t.getLastColumn());

    if (t.getLastRow() > 1 && cols > 1) {
      // Remove existing charts
      d.getCharts().forEach(chart => d.removeChart(chart));
```

```
// Chart 1: First 4 symbols
const chartCols1 = Math.min(5, cols); // Timestamp + 4 symbols
const c1 = d.newChart()
  .setChartType(Charts.ChartType.LINE)
  .addRange(d.getRange(7, 1, 350, chartCols1))
  .setOption('title', 'Primary Symbols - Price Trend')
  .setOption('legend', { position: 'bottom' })
  .setOption('hAxis', { title: 'Time' })
  .setOption('vAxis', { title: 'Price' })
  .setOption('width', 800)
  .setOption('height', 400)
  .setPosition(10, 1, 0, 0)
  .build();
d.insertChart(c1);

// Chart 2: All symbols
if (cols > 5) {
  const c2 = d.newChart()
    .setChartType(Charts.ChartType.LINE)
    .addRange(d.getRange(7, 1, 350, Math.min(12, cols)))
    .setOption('title', 'All Symbols - Price Trend')
    .setOption('legend', { position: 'bottom' })
    .setOption('hAxis', { title: 'Time' })
    .setOption('vAxis', { title: 'Price' })
    .setOption('width', 800)
    .setOption('height', 400)
    .setPosition(10, 10, 0, 0)
    .build();
  d.insertChart(c2);
}
```

```
    }

    console.log('Dashboard built successfully');
  } catch (err) {
    logError_('BuildDashboard_', err);
  }
}

/* ---------- Menu + Triggers ---------- */
function onOpen() {
  try {
    SpreadsheetApp.getUi().createMenu('TradingOps')
      .addItem('⚡ QuickStart (First Time Setup)', 'QuickStart')
      .addSeparator()
      .addItem('📊 Update Prices Now', 'UpdateTicks_')
      .addItem('📈 Rebuild Dashboard', 'BuildDashboard_')
      .addItem('🔄 Install Poll Trigger', 'InstallTrigger_')
      .addItem('🛑 Remove Poll Trigger', 'RemoveTrigger_')
      .addSeparator()
      .addItem('🧹 Cleanup Old Data', 'CleanupOldData_')
      .addItem('📋 Export to CSV', 'ExportToCSV_')
      .addItem('❓ Show Help', 'ShowHelp_')
      .addToUi();
  } catch (err) {
    console.error('Error creating menu:', err);
  }
}

function InstallTrigger_() {
  try {
    const mins = Math.max(1, parseInt(cfg_('POLL_INTERVAL_MIN') || '1', 10));
```

```
    // Validate interval (Apps Script limits)
    const validIntervals = [1, 5, 10, 15, 30];
    const interval = validIntervals.includes(mins) ? mins : 1;

    // Remove existing triggers
    RemoveTrigger_();

    // Create new trigger
    ScriptApp.newTrigger('UpdateTicks_')
      .timeBased()
      .everyMinutes(interval)
      .create();

    SpreadsheetApp.getActiveSpreadsheet().toast(
      `Price polling trigger installed! Updates every ${interval} minute(s).`,
      'Success',
      5
    );

    setHL_('Polling Interval', `${interval} minute(s)`);
  } catch (err) {
    logError_('InstallTrigger_', err);
    SpreadsheetApp.getUi().alert('Failed to install trigger: ' + err.message);
  }
}

function RemoveTrigger_() {
  try {
    const triggers = ScriptApp.getProjectTriggers();
    let removed = 0;
```

```
    triggers.forEach(t => {
      if (t.getHandlerFunction() === 'UpdateTicks_') {
        ScriptApp.deleteTrigger(t);
        removed++;
      }
    });

    if (removed > 0) {
      SpreadsheetApp.getActiveSpreadsheet().toast(
        `Removed ${removed} polling trigger(s).`,
        'Success',
        3
      );
    }
  } catch (err) {
    logError_('RemoveTrigger_', err);
  }
}

function QuickStart() {
  try {
    const ui = SpreadsheetApp.getUi();
    const response = ui.alert(
      'TradingOps QuickStart',
      'This will initialize all sheets and start price polling. Continue?',
      ui.ButtonSet.YES_NO
    );

    if (response !== ui.Button.YES) {
      return;
```

```
    }

    // Show progress
    SpreadsheetApp.getActiveSpreadsheet().toast('Setting up...', 'QuickStart', -1);

    Bootstrap_();          // Create tabs + defaults
    Utilities.sleep(1000); // Brief pause
    UpdateTicks_();        // Pull prices immediately
    Utilities.sleep(1000);
    BuildDashboard_();     // Create charts
    Utilities.sleep(1000);
    InstallTrigger_();     // Start polling

    SpreadsheetApp.getActiveSpreadsheet().toast('Setup complete!', 'Success', 5);

    // Show welcome message
    ui.alert(
      'Setup Complete!',
      'TradingOps is now running.\\n\\n' +
      '✓ Sheets initialized\\n' +
      '✓ First price update complete\\n' +
      '✓ Dashboard created\\n' +
      '✓ Auto-polling enabled\\n\\n' +
      'Check the Dashboard tab to see your charts!',
      ui.ButtonSet.OK
    );
  } catch (err) {
    logError_('QuickStart', err);
    SpreadsheetApp.getUi().alert('QuickStart failed: ' + err.message);
  }
}
```

```
function CleanupOldData_() {
  try {
    const ui = SpreadsheetApp.getUi();
    const response = ui.prompt(
      'Cleanup Old Data',
      'Keep how many recent rows in Ticks sheet? (Enter number)',
      ui.ButtonSet.OK_CANCEL
    );

    if (response.getSelectedButton() !== ui.Button.OK) {
      return;
    }

    const keepRows = parseInt(response.getResponseText(), 10);
    if (isNaN(keepRows) || keepRows < 1) {
      ui.alert('Invalid number entered.');
      return;
    }

    const t = sh_(SH.TICKS);
    const totalRows = t.getLastRow();

    if (totalRows <= keepRows + 1) {
      ui.alert('No cleanup needed. Current rows: ' + (totalRows - 1));
      return;
    }

    const deleteCount = totalRows - keepRows - 1;
    t.deleteRows(2, deleteCount);
```

```
    ui.alert(`Cleanup complete! Removed ${deleteCount} old rows.`);
  } catch (err) {
    logError_('CleanupOldData_', err);
    SpreadsheetApp.getUi().alert('Cleanup failed: ' + err.message);
  }
}

function ExportToCSV_() {
  try {
    const ui = SpreadsheetApp.getUi();
    ui.alert(
      'Export to CSV',
      'To export data:\\n\\n' +
      '1. Go to the Ticks sheet\\n' +
      '2. Click File → Download → CSV (.csv)\\n\\n' +
      'This will export the current sheet as CSV.',
      ui.ButtonSet.OK
    );
  } catch (err) {
    logError_('ExportToCSV_', err);
  }
}

function ShowHelp_() {
  try {
    const ui = SpreadsheetApp.getUi();
    ui.alert(
      'TradingOps Help',
      'SHEETS:\\n' +
      '• Config: API keys and settings\\n' +
      '• Watchlist: Symbols to track (set Active=FALSE to disable)\\n' +
```

```
    '• Ticks: Historical price data\\n' +
    '• Alerts: Webhook notifications\\n' +
    '• Health: System status\\n' +
    '• Dashboard: Charts and summaries\\n\\n' +
    'MENU ITEMS:\\n' +
    '• QuickStart: Initial setup (run once)\\n' +
    '• Update Prices Now: Manual price fetch\\n' +
    '• Install/Remove Poll Trigger: Auto-updates\\n' +
    '• Cleanup Old Data: Remove old ticks\\n\\n' +
    'For webhook setup:\\n' +
    '1. Deploy as Web App (Extensions → Apps Script)\\n' +
    '2. Copy URL to Health sheet\\n' +
    '3. Use in TradingView or other platforms',
    ui.ButtonSet.OK
   );
  } catch (err) {
   logError_('ShowHelp_', err);
  }
}
```