NYSE OPEN > IDN TERMINAL v26.5 > SECURE SESSION > EDGE-GATED

Programming Manual

Script Lab - Building Indicators & Strategies

IntelliData Network Trading & Intelligence Terminal
Version 26.5 · Edition 1
Issued for licensed clients

CONTENTS

01 · Introduction to Script Lab

Script Lab is the IntelliData Terminal's built-in programming environment for creating your own technical **indicators** and automated **trading strategies**. You write short JavaScript programs against a clean, purpose-built market API; the terminal runs them over the active symbol's price history, plots your output, and - for strategies - simulates the trades and reports performance.

You do not need to be a professional developer. If you can read the worked examples in this manual and change a number, you can build something useful. This guide covers every function in the API and walks through five complete examples you can adapt.

> **Safe by design.** Your code runs in an isolated background worker with no access to the page, your account, or the network. A script that runs too long (more than 8 seconds) is stopped automatically, so a mistake such as an endless loop can never freeze the terminal.

02 · Opening Script Lab

Load the terminal, then open the main panel's **SCRIPT** view tab (the title bar reads *IDN SCRIPT LAB - INDICATOR & STRATEGY EDITOR*). The editor opens with a working EMA-crossover example already loaded so you can run something immediately.

03 · The Script Lab Workspace

The workspace has a toolbar, a code editor on the left, and a results area on the right (a preview chart, a statistics strip, and a console). The toolbar controls are:

Control	Purpose
Language	JavaScript is active today; Python is marked "soon."
Name	A name for your script, used when you save it.
Examples	Load one of the built-in example scripts.
Saved	Reload a script you previously saved in this browser.
■ RUN	Execute the script and plot it on the preview chart.
■ BACKTEST	Run a strategy and compute performance statistics.
■ + CHART	Pin this indicator onto the main CHART tab.
SAVE	Save the script to this browser.
NEW	Start a fresh, empty script.
API	Toggle the built-in ctx API reference.
■ POP-OUT	Open Script Lab in its own resizable window.

The **console** shows anything you print with `ctx.log(...)` and any error messages. The **statistics strip** fills in after a backtest.

04 · Two Kinds of Script: Indicator vs Strategy

Every Script Lab program is one of two shapes, decided by which functions you define:

Indicator - draw something

Define a function named `indicator(ctx)`. It runs once, sees the whole history, and uses `ctx.plot(...)` to draw lines. Indicators never trade. Use them for moving averages, bands, oscillators, and overlays.

Strategy - trade something

Define `onBar(ctx)` (and optionally `onInit(ctx)`). The engine walks through history one bar at a time and calls `onBar` on each, where you place entry and exit orders. Run it with **BACKTEST** to simulate the trades and measure results.

You define	Run with	The engine...
`indicator(ctx)`	RUN	executes once at the latest bar and collects your plots
`onInit + onBar`	BACKTEST	loops every bar calling onBar, simulates fills, then computes stats

> A script may define both: use `onInit` to set up and plot your indicators, and `onBar` to trade off them. The EMA-cross example does exactly this.

05 · The ctx Object

Your functions receive one argument, conventionally named `ctx` ("context"). It is your entire toolbox: the price data, the indicator math, the plotting and logging functions, the trading commands, and helpful constants. Everything in this manual is reached through `ctx` - for example `ctx.close`, `ctx.ema(...)`, `ctx.enterLong(...)`.

You may also stash your own values on `ctx` to share them between `onInit` and `onBar`. The examples store indicators this way, e.g. `ctx.fast = ctx.ema(ctx.close, 9)` in onInit, then read `ctx.fast` in onBar.

06 · Series & Reverse Indexing

Price data and indicator results are **Series**. The single most important rule in Script Lab is how a Series is indexed:

> **Series are reverse-indexed.** Index `[0]` is the current bar, `[1]` is one bar ago, `[2]` two bars ago, and so on - the same convention used by Pine and NinjaScript.

```
close[0]   // the current bar's close
close[1]   // the previous bar's close
high[2]    // the high two bars ago

// "is this close higher than the last one?"
if (close[0] > close[1]) { /* rising */ }
```

In a **strategy**, [0] means the bar currently being processed as the engine walks forward. In an **indicator**, [0] is the most recent bar. Each Series also exposes `.length` and `.raw` (the underlying oldest-to-newest array),

which is handy when you want to loop over all history - the Bollinger example uses `.raw`.

07 · Writing Your First Indicator

This plots two moving averages over price. Select RUN to see them on the preview chart.

```
function indicator(ctx){
  var fast = ctx.ema(ctx.close, 9);
  var slow = ctx.ema(ctx.close, 21);
  ctx.plot(fast, "EMA9",  ctx.color.yellow);
  ctx.plot(slow, "EMA21", ctx.color.blue);
}
```

`ctx.ema` returns a Series; `ctx.plot` draws it with a label and color. That is a complete, valid indicator.

08 · Writing Your First Strategy

A strategy adds `onBar`, where you act on each bar. This buys when the fast average crosses above the slow one and exits when it crosses back below.

```
function onInit(ctx){
  ctx.fast = ctx.ema(ctx.close, 9);
  ctx.slow = ctx.ema(ctx.close, 21);
  ctx.plot(ctx.fast, "EMA9",  ctx.color.yellow);
  ctx.plot(ctx.slow, "EMA21", ctx.color.blue);
}

function onBar(ctx){
  if (ctx.crossover(ctx.fast, ctx.slow) && ctx.position.flat)
    ctx.enterLong({ qty: 100 });
  if (ctx.crossunder(ctx.fast, ctx.slow) && ctx.position.long)
    ctx.exitLong();
}
```

Press **BACKTEST**. The engine runs the strategy across history, marks entries and exits on the chart, and fills the statistics strip with net profit, win rate, and more.

09 · Plotting

Draw a Series with:

```
ctx.plot(series, name, color, pane)
```

- **series** - a Series (or plain array) of values to draw.
- **name** - a label shown in the legend.
- **color** - use the `ctx.color` palette (below) or any CSS color string.
- **pane** - `ctx.pane.main` overlays on price; `ctx.pane.lower` draws in a separate panel beneath (use this for oscillators like RSI and MACD).

Color palette

`ctx.color` provides: yellow, blue, amber, purple, green, red, white, gray.

```
ctx.plot(ctx.rsi(ctx.close,14), "RSI", ctx.color.purple, ctx.pane.lower);
```

10 · Built-in Data Series

These Series are always available on `ctx`, drawn from the active symbol's price history:

Series	Meaning
`ctx.open`	Bar opening prices
`ctx.high`	Bar highs
`ctx.low`	Bar lows
`ctx.close`	Bar closes (most common source)
`ctx.volume`	Bar volume
`ctx.time`	Bar timestamps (unix seconds)

11 · Indicator Function Reference

Each function returns a Series (except `macd`, which returns three). Pass a source Series such as `ctx.close` and a length.

Function	Returns / Notes
`ema(src, len)`	Exponential moving average
`sma(src, len)`	Simple moving average
`rma(src, len)`	Wilder's smoothing (used inside RSI/ATR)
`rsi(src, len)`	Relative Strength Index (len defaults to 14)
`atr(len)`	Average True Range (len defaults to 14)
`stdev(src, len)`	Standard deviation over len
`highest(src, len)`	Highest value over the last len bars
`lowest(src, len)`	Lowest value over the last len bars
`macd(src, fast, slow, sig)`	Object { macd, signal, hist } - defaults 12/26/9
`change(src, k)`	Difference src[i] minus src[i-k] (k defaults to 1)

12 · Signal Helpers

These return true exactly on the bar where a cross happens, so they are ideal inside `onBar`. The second argument may be another Series or a fixed number.

Helper	True when...
`crossover(a, b)`	a crosses from at-or-below b to above b
`crossunder(a, b)`	a crosses from at-or-above b to below b

```
var ma50 = ctx.ema(ctx.close, 50);
if (ctx.crossover(ctx.close, ma50))
  ctx.log("price crossed above the 50");
if (ctx.crossunder(ctx.rsi(ctx.close,14), 70))
  ctx.log("RSI fell back under 70");
```

13 · The Trading API

Available in strategies (inside onBar). Orders fill at the current bar's close by default; pass `{ price: ... }` to override, and `{ qty: ... }` to set size (otherwise the default size is used).

Order	Effect
enterLong({ qty })	Open or flip to a long position
enterShort({ qty })	Open or flip to a short position
exitLong()	Close an open long
exitShort()	Close an open short
closeAll()	Flatten any open position

You may also tag an order with `{ tag: "my-reason" }`; the tag is recorded on the resulting trade for review.

> Entering long while short (or vice-versa) automatically closes the existing position and reverses - you do not need to flatten first.

14 · The Position Object

`ctx.position` tells you your current state. Read it to avoid stacking orders or to add exit logic.

Property	Meaning
position.flat	true when you hold nothing
position.long	true when net long
position.short	true when net short
position.size	signed position size (+long / -short)
position.side	'long', 'short', or 'flat'
position.avgPrice	average entry price of the open position

```
if (ctx.position.flat && ctx.crossover(ctx.fast, ctx.slow))
  ctx.enterLong({ qty: 100 });
// simple 3% stop
if (ctx.position.long &&
    ctx.close[0] < ctx.position.avgPrice * 0.97)
  ctx.exitLong();
```

15 · Context Properties & Helpers

Property	Meaning
`ctx.i`	Index of the bar currently being processed
`ctx.bars / ctx.n`	Total number of bars in the history
`ctx.na(x)`	true if x is null or not a finite number
`ctx.nan`	the null/empty value
`ctx.color`	color palette object (see Plotting)
`ctx.pane`	{ main, lower } pane selectors
`ctx.log(...)`	print values to the console

Use `ctx.na()` to guard against the early bars of a Series, which are empty until the indicator has enough data (for example, a 21-period EMA has no value for the first 20 bars).

16 · Complete ctx API Reference

The full surface, identical to the in-app API panel (click **API** in the toolbar).

Member	Description
`open / high / low / close`	Price Series (reverse-indexed; close[0]=current)
`volume / time`	Volume and unix-time Series
`sma(src,len) / ema(src,len)`	Simple / exponential moving average
`rma(src,len)`	Wilder moving average
`rsi(src,len) / atr(len)`	RSI / Average True Range
`stdev(src,len)`	Standard deviation
`highest(src,len) / lowest(src,len)`	Rolling max / min
`macd(src,fast,slow,sig)`	Returns { macd, signal, hist }
`change(src,k)`	src[i] - src[i-k]
`crossover(a,b) / crossunder(a,b)`	Cross detection
`plot(series,name,color,pane)`	Plot a Series
`log(...args)`	Print to console
`enterLong / enterShort({qty})`	Open or flip position
`exitLong / exitShort / closeAll`	Close positions
`position`	{ size, side, avgPrice, flat, long, short }
`i / bars`	Current bar index / total bars
`color / pane`	Palette and pane selectors
`na(x) / nan`	Null checking helpers

17 · Backtesting & Performance Stats

When you BACKTEST a strategy, the engine starts from your configured starting capital, processes every bar, simulates fills, force-closes any open position at the end, and reports:

Statistic	Meaning
Net Profit	Total profit/loss across all trades
Return %	Net profit as a percent of starting capital
Total Trades	Number of closed trades
Wins / Losses	Count of profitable vs losing trades
Win Rate	Percentage of trades that were profitable
Profit Factor	Gross profit divided by gross loss
Max Drawdown	Largest peak-to-trough drop in account value
Final Equity	Account value at the end of the test

Starting capital and default order size come from the terminal's Settings (paper-trading section). The preview chart marks each entry and exit so you can see where the strategy acted.

18 · Plotting onto the Main Chart

An indicator you like in Script Lab can be pinned onto the terminal's main price chart. With your script loaded, click **+ CHART**. It is added as a named *study* and appears on the CHART tab; a chip with its name shows on the Script Lab studies row, with an × to remove it.

Pinned studies are remembered in your browser, so they persist between sessions until you remove them. This is how you build a personal set of overlays that travel with you on every symbol.

19 · Saving, Loading & Examples

- **SAVE** stores the current script (under its Name) in this browser; reload it later from the **Saved** menu.
- **NEW** clears the editor to start fresh.
- **Examples** loads any of the five built-in templates - the fastest way to learn by editing working code.
- Scripts and pinned studies live in your browser's local storage; they are private to you and to this machine.

> **Back up important scripts.** Because scripts are stored in the browser, clearing site data will remove them. Keep a copy of anything valuable in a text file.

20 · Worked Examples

All five ship under the Examples menu. Read them top to bottom - they introduce every concept in this manual.

EMA Crossover (strategy)

Trend-following: long when the fast EMA crosses above the slow, flat when it crosses back.

```
function onInit(ctx){
  ctx.fast = ctx.ema(ctx.close, 9);
  ctx.slow = ctx.ema(ctx.close, 21);
  ctx.plot(ctx.fast, "EMA9",  ctx.color.yellow);
  ctx.plot(ctx.slow, "EMA21", ctx.color.blue);
}
function onBar(ctx){
  if (ctx.crossover(ctx.fast, ctx.slow) && ctx.position.flat)
    ctx.enterLong({ qty: 100 });
  if (ctx.crossunder(ctx.fast, ctx.slow) && ctx.position.long)
    ctx.exitLong();
}
```

RSI Mean Reversion (strategy)

Buy as RSI climbs back above 30 (oversold), sell as it drops back below 70 (overbought). Note RSI is plotted in the lower pane.

```
function onInit(ctx){
  ctx.r = ctx.rsi(ctx.close, 14);
  ctx.plot(ctx.r, "RSI", ctx.color.purple, ctx.pane.lower);
}
function onBar(ctx){
  if (ctx.r[1] <= 30 && ctx.r[0] > 30 && ctx.position.flat)
    ctx.enterLong({ qty:100 });
  if (ctx.r[1] >= 70 && ctx.r[0] < 70 && ctx.position.long)
    ctx.exitLong();
}
```

Bollinger Bands (indicator)

A 20-period basis with bands two standard deviations away. Shows looping over `.raw` to build derived arrays.

```
function indicator(ctx){
  var basis = ctx.sma(ctx.close, 20);
  var dev   = ctx.stdev(ctx.close, 20);
  var up=[], dn=[]; var bs=basis.raw, d=dev.raw;
  for (var i=0;i<ctx.bars;i++){
    up.push(bs[i]==null ? null : bs[i] + 2*d[i]);
    dn.push(bs[i]==null ? null : bs[i] - 2*d[i]);
  }
  ctx.plot(basis, "Basis", ctx.color.amber);
  ctx.plot(up, "Upper", ctx.color.blue);
  ctx.plot(dn, "Lower", ctx.color.blue);
}
```

MACD Histogram (indicator)

Classic MACD in the lower pane, using the three Series returned by `macd`.

```
function indicator(ctx){
  var m = ctx.macd(ctx.close, 12, 26, 9);
  ctx.plot(m.macd,   "MACD",   ctx.color.blue,  ctx.pane.lower);
  ctx.plot(m.signal, "Signal", ctx.color.amber, ctx.pane.lower);
  ctx.plot(m.hist,   "Hist",   ctx.color.gray,  ctx.pane.lower);
}
```

ATR Breakout (strategy)

Enter when price closes above the prior 20-bar high; exit on an ATR-based trailing stop.

```
function onInit(ctx){
  ctx.hh = ctx.highest(ctx.high, 20);
  ctx.a  = ctx.atr(14);
  ctx.plot(ctx.hh, "20H", ctx.color.gray);
}
function onBar(ctx){
  if (ctx.position.flat && ctx.close[0] > ctx.hh[1])
    ctx.enterLong({ qty:100 });
  else if (ctx.position.long &&
           ctx.close[0] < ctx.close[1] - 2*ctx.a[0])
    ctx.exitLong();
}
```

21 · Best Practices & Gotchas

- **Remember reverse indexing.** [0] is now, [1] is the prior bar. Comparing [0] to [1] is how you detect change.
- **Guard the early bars.** Indicators are empty until they have enough data; check with `ctx.na(x)` before using a value.
- **Check position before ordering.** Use `ctx.position.flat/long/short` so you do not stack or fight your own orders.
- **Indicators plot, strategies trade.** If you define onBar, run it with BACKTEST - RUN alone will remind you to backtest.
- **Fills are at the close** of the signal bar unless you pass a price - keep expectations realistic.
- **Avoid heavy loops.** Scripts are stopped after 8 seconds; vectorized indicator calls are far faster than deep nested loops.
- **Back up scripts** you care about; they live only in this browser.

22 · Quick Reference

Program shapes

```
indicator(ctx){...}            // RUN  -> plots only
onInit(ctx){...} onBar(ctx){...} // BACKTEST -> trades + stats
```

Data

```
ctx.open  ctx.high  ctx.low  ctx.close  ctx.volume  ctx.time
// reverse-indexed: close[0]=now, close[1]=prior
```

Indicators

```
ema sma rma rsi atr stdev highest lowest macd change
```

Signals & trading

```
ctx.crossover(a,b)  ctx.crossunder(a,b)
ctx.enterLong({qty})  ctx.enterShort({qty})
ctx.exitLong()  ctx.exitShort()  ctx.closeAll()
ctx.position.flat / .long / .short / .size / .avgPrice
```

Plot & log

```
ctx.plot(series,name,ctx.color.blue,ctx.pane.lower)   ctx.log(...)
```

For everything else in the terminal - charts, news, alerts, paper trading - see the **Terminal Operating Manual** in the MANUALS menu.