IB Gateway Futures Charting: The Missing Pieces Supplement

This supplement fills the critical operational gaps that documentation alone cannot convey. **Paper trading accounts require shared market data subscriptions from live accounts** Interactive Brokers (a 24-hour propagation delay applies), Interactive Brokers historical data requests **always require subscriptions regardless of delayed mode**, GitHub and IB Gateway should be restarted daily at **11:45 PM ET** to prevent memory leaks and connection issues. The micro futures MNQ, MES, and MGC all require the CME Real-Time subscription bundle (~$10-15/month non-professional), and you must complete the **Market Data API Acknowledgement** form in Client Portal or all API data requests will fail with "not subscribed" errors even with valid subscriptions. Interactive Brokers

---

IB Gateway configuration essentials

Exact API settings and menu paths

TWS: File/Edit → Global Configuration → API → Settings (File menu for Mosaic, Edit for Classic) **IB Gateway**: Configure (gear icon) → Settings → API → Settings

Setting	Default	Production Recommendation
Enable ActiveX and Socket Clients	OFF (TWS), ON (Gateway)	**Must be ENABLED**
Read-Only API	ON	Disable if placing orders
Create API message log file	OFF	Enable for debugging
Socket port	7496/7497 (TWS), 4001/4002 (Gateway)	Keep defaults

Port configuration decision matrix

Scenario	Port	Notes
IB Gateway + Live Account	**4001**	Production default
IB Gateway + Paper Account	**4002**	Testing/development
TWS + Live Account	**7496**	When GUI monitoring needed
TWS + Paper Account	**7497**	Development with visual feedback

Critical insight: Ports are configurable but client port must match exactly. Multiple TWS instances require different ports. GitHub Interactive Brokers IB API is **unencrypted TCP—**use SSH tunnel for remote access. Docker Hub

Client ID management strategy

Client IDs distinguish multiple API applications connecting to the same Gateway session. GitHub **Maximum 32 simultaneous clients** per session.

```
Recommended allocation:
├── 0: Reserved (receives TWS GUI orders)
├── 1-10: Primary trading strategies
├── 11-20: Market data collection/charting
├── 21-30: Analysis applications
└── 31: Testing/debugging
```

Duplicate client ID behavior: Error code **507** (Java) or **-1** (C#) thrown; second connection rejected, first remains active.

Market data subscription verification

Navigate to **Client Portal → Settings → Account Settings → Market Data Subscriptions**. For MNQ, MGC, and MES, you need:

Product	Required Subscription
MES, MNQ, M2K	CME Real-Time (Level 1)
MGC	COMEX (included in CME bundle)

Hidden requirement: Complete the **Market Data API Acknowledgement** form at Client Portal → Settings → User Settings → Market Data API Access. Without this form completed, API data requests return "not subscribed" errors even with active subscriptions. Interactive Brokers

Error code 354 ("Requested market data is not subscribed") indicates missing subscription or incomplete API acknowledgement form. GitHub

IB Gateway versus TWS for production

IB Gateway uses **~40% less memory**, has faster startup, and is designed for API-only headless operation. github TWS provides visual feedback useful for debugging.

Critical constraint: Same username cannot run both simultaneously. Create additional users in Account Management if you need both. Market data subscriptions are per-username.

Recommendation: Use IB Gateway for production servers; use TWS locally for development where visual monitoring helps.

Error handling and recovery patterns

Connection state error codes

These are the most important codes to handle for connection resilience: Interactive Brokers

Code	Meaning	Required Action
1100	IB server connection lost	Block new requests; wait for 1101/1102
1101	Connection restored, data lost	**Re-subscribe all market data immediately**
1102	Connection restored, data maintained	Resume normal operation
2103	Market data farm disconnected	Wait; usually temporary during nightly reset
2105	Historical data farm disconnected	Historical requests will fail
2104, 2106, 2158	Farm connection OK	**Not errors**—informational only

Historical data pacing rules

For bars ≤30 seconds, IB enforces strict pacing:

1. **15-second rule**: Wait 15 seconds between identical historical data requests
2. **6-per-2-seconds rule**: Maximum 6 requests per 2 seconds for same contract/exchange/tick type
3. **60-in-10-minutes rule**: Maximum 60 requests in any 10-minute rolling window
4. **BID_ASK requests count double**: Each counts as 2 requests Interactive Brokers

```python
```

```python
# Pacing-compliant request checker
async def can_send_historical_request(req, last_requests):
    now = time.time()

    # Rule 1: No identical requests within 15 seconds
    req_hash = f"{req.contract.conId}:{req.end_datetime}:{req.bar_size}"
    if req_hash in last_requests and now - last_requests[req_hash] < 15:
        return False

    # Rule 2: Max 6 requests per contract in 2 seconds
    contract_requests = [t for t in contract_times.get(req.contract.conId, [])
                         if now - t < 2]
    if len(contract_requests) >= 6:
        return False

    # Rule 3: Max 60 in 10 minutes globally
    global_requests = [t for t in all_request_times if now - t < 600]
    if len(global_requests) >= 60:
        return False

    return True
```

Exponential backoff implementation

python

```python
RETRYABLE_ERRORS = {162, 366, 1100, 2103, 2105}  # Pacing, connection issues
FATAL_ERRORS = {200, 354, 502}  # Bad contract, no subscription, can't connect

async def ib_request_with_retry(func, *args, max_retries=5, base_delay=2.0):
    for attempt in range(max_retries + 1):
        try:
            return await func(*args)
        except IBError as e:
            if e.code in FATAL_ERRORS:
                raise  # Don't retry
            if e.code not in RETRYABLE_ERRORS or attempt == max_retries:
                raise

            delay = min(base_delay * (2 ** attempt), 60.0)
            delay *= (0.75 + random.random() * 0.5)  # Jitter

            # Special handling for pacing violations
            if e.code == 162 and "pacing violation" in str(e.message).lower():
                delay = max(delay, 15.0)

            await asyncio.sleep(delay)
```

Connection recovery state machine

python

```python
class ConnectionState(Enum):
    DISCONNECTED = 0
    CONNECTING = 1
    CONNECTED = 2       # Socket connected
    READY = 3           # Data farms OK
    SUBSCRIBED = 4      # Active market data
    IB_DISCONNECTED = 5  # 1100 received

class IBConnectionManager:
    def __init__(self, host, port, client_id):
        self.ib = IB()
        self.state = ConnectionState.DISCONNECTED
        self.ib_server_connected = True
        self.active_subscriptions = []

    def _on_error(self, reqId, errorCode, errorString, contract):
        if errorCode == 1100:
            self.ib_server_connected = False
            self.state = ConnectionState.IB_DISCONNECTED
        elif errorCode == 1101:
            self.ib_server_connected = True
            self.state = ConnectionState.CONNECTED
            self._resubscribe_all()  # Critical: must re-subscribe
        elif errorCode == 1102:
            self.ib_server_connected = True
            if self.state == ConnectionState.IB_DISCONNECTED:
                self.state = ConnectionState.SUBSCRIBED
```

Futures contract rolling and continuous data

ContFuture capabilities and limitations

IB's `CONTFUT` security type provides automatically-stitched historical data using **ratio adjustment**:

```python
```

```python
from ib_insync import ContFuture

mnq_cont = ContFuture('MNQ', 'CME', currency='USD')
mes_cont = ContFuture('MES', 'CME', currency='USD')
mgc_cont = ContFuture('MGC', 'COMEX', currency='USD')

# Historical data request - endDateTime MUST be empty
bars = ib.reqHistoricalData(
    mnq_cont,
    endDateTime='',  # Required for ContFuture
    durationStr='1 Y',
    barSizeSetting='1 day',
    whatToShow='TRADES',
    useRTH=True
)
```

Critical limitations of ContFuture:

- Historical data **only**—cannot use for real-time streaming or orders
- Roll timing determined by IB, not customizable
- Ratio-adjusted prices may not match your strategy requirements

Use Case	ContFuture	Specific Contract
Historical backtesting	✓	
Technical analysis spanning expirations	✓	
Live trading/real-time data		✓
Custom roll timing		✓

Contract expiration detection

```python
```

```python
from datetime import datetime, timedelta
from ib_insync import Future

def get_contracts_for_rolling(ib, symbol, exchange='CME'):
    """Get front month and next contract for roll management"""
    generic = Future(symbol=symbol, exchange=exchange, currency='USD')
    details = ib.reqContractDetails(generic)

    today = datetime.now().strftime('%Y%m%d')
    active = [d for d in details
              if d.contract.lastTradeDateOrContractMonth >= today]
    active.sort(key=lambda x: x.contract.lastTradeDateOrContractMonth)

    front_month = active[0].contract if active else None
    back_month = active[1].contract if len(active) > 1 else None

    return front_month, back_month

def should_roll(contract_details, days_before=8):
    """Check if roll should occur (calendar-based)"""
    last_trade = datetime.strptime(
        contract_details.contract.lastTradeDateOrContractMonth, '%Y%m%d'
    )
    roll_date = last_trade - timedelta(days=days_before)
    return datetime.now() >= roll_date
```

MNQ, MES, MGC roll schedule 2025-2026

These quarterly contracts (H=March, M=June, U=September, Z=December) have consistent patterns:

Contract	Expiration (3rd Friday)	Official Roll Thursday	Recommended Roll
H25 (Mar)	Mar 21, 2025	Mar 13, 2025	Mar 11-12, 2025
M25 (Jun)	Jun 20, 2025	Jun 12, 2025	Jun 10-11, 2025
U25 (Sep)	Sep 19, 2025	Sep 11, 2025	Sep 9-10, 2025
Z25 (Dec)	Dec 19, 2025	Dec 11, 2025	Dec 9-10, 2025
H26 (Mar)	Mar 20, 2026	Mar 12, 2026	Mar 10-11, 2026
M26 (Jun)	Jun 19, 2026	Jun 11, 2026	Jun 9-10, 2026

Industry standard: Roll **8-10 days before expiration**. Volume typically migrates to back month by the second Thursday of expiration month at 9:30 AM ET.

Historical data stitching for custom continuous contracts

```python
def build_continuous_series(ib, symbol, contracts_data, roll_dates):
    """
    Backwards Panama adjustment: current price is 'true',
    historical prices adjusted by absolute difference at roll points
    """
    adjusted_data = []
    cumulative_adjustment = 0

    # Start from most recent, work backwards
    for i, (contract, data) in enumerate(contracts_data):
        if i == 0:
            adjusted_data.append(data)  # Most recent—no adjustment
        else:
            roll_date = roll_dates[i-1]
            new_price = contracts_data[i-1][1].loc[roll_date, 'close']
            old_price = data.loc[roll_date, 'close']

            adjustment = new_price - old_price
            cumulative_adjustment += adjustment

            data_adjusted = data.copy()
            data_adjusted[['open', 'high', 'low', 'close']] += cumulative_adjustment
            adjusted_data.append(data_adjusted)

    return pd.concat(adjusted_data).sort_index()
```

Requesting expired contract data

```python
```

```
# includeExpired=True is critical for expired contracts
expired_contract = Future(
    symbol='MNQ',
    lastTradeDateOrContractMonth='202409',  # September 2024
    exchange='CME',
    currency='USD',
    includeExpired=True  # CRITICAL
)

ib.qualifyContracts(expired_contract)
bars = ib.reqHistoricalData(expired_contract, ...)
```

Limitation: Historical data for expired futures available up to **2 years from expiration date**. Interactive Brokers

---

Development and testing setup

Paper trading data access reality

Paper accounts do NOT receive market data by default. To enable:

1. Client Portal → Settings → Paper Trading Account
2. Enable **"Share real-time market data"**
3. Wait **up to 24 hours** for propagation

Critical constraint: Live and paper accounts **cannot receive market data simultaneously**—data goes to whichever is logged in. You cannot run both with real-time data.

What works without subscriptions

Function	Without Subscription	Notes
reqContractDetails()	✅ Works 24/7	Always available
reqHistoricalData()	❌ **Requires subscription**	No delayed mode workaround
reqMktData() with delayed	✅ 15-minute delay	Use ib.reqMarketDataType(3)
reqMarketDataType(4) frozen	✅ Last known values	Works when markets closed

Delayed mode does NOT apply to historical data—this is a common misconception. Historical bars always require an active subscription. Interactive Brokers

Docker containers for IB Gateway

The gnzsnz/ib-gateway image is the most maintained option: GitHub

```yaml
services:
  ib-gateway:
    image: ghcr.io/gnzsnz/ib-gateway:stable
    restart: unless-stopped
    environment:
      TWS_USERID: ${TWS_USERID}
      TWS_PASSWORD: ${TWS_PASSWORD}
      TRADING_MODE: paper
      AUTO_RESTART_TIME: "11:45 PM"
      TWOFA_TIMEOUT_ACTION: restart
      JAVA_HEAP_SIZE: "1024"
    ports:
      - "127.0.0.1:4001:4003"  # Live
      - "127.0.0.1:4002:4004"  # Paper
    healthcheck:
      test: ["CMD-SHELL", "pgrep java && exit 0"]
      interval: 10s
```

Mocking for unit tests

```python
```

```
from unittest.mock import Mock, patch

class TestChartingStrategy(unittest.TestCase):
    def setUp(self):
        self.mock_ib = Mock()
        self.mock_ib.isConnected.return_value = True

    def test_historical_data_request(self):
        mock_bars = [
            Mock(date='2025-01-01', open=100, high=101, low=99, close=100.5),
            Mock(date='2025-01-02', open=100.5, high=102, low=100, close=101),
        ]
        self.mock_ib.reqHistoricalData.return_value = mock_bars

        result = your_strategy.get_bars(self.mock_ib, 'MES')
        self.assertEqual(len(result), 2)
```

Building a data cache for offline development

```
python

class IBDataCache:
    def __init__(self, cache_dir='./ib_cache'):
        self.cache_dir = Path(cache_dir)
        self.cache_dir.mkdir(exist_ok=True)

    def cache_historical_data(self, contract, bars):
        key = f"{contract.symbol}_{contract.lastTradeDateOrContractMonth}"
        path = self.cache_dir / f"hist_{key}.csv"
        pd.DataFrame([b.__dict__ for b in bars]).to_csv(path, index=False)

    def load_cached_data(self, contract):
        key = f"{contract.symbol}_{contract.lastTradeDateOrContractMonth}"
        path = self.cache_dir / f"hist_{key}.csv"
        return pd.read_csv(path) if path.exists() else None
```

Production deployment considerations

Daily restart is non-negotiable

IB Gateway has documented memory leaks with high-frequency data requests. **Configure auto-restart at**

11:45 PM ET (before IB's midnight server reset):

```ini
# IBC config.ini
AutoRestartTime=11:45 PM
TWOFA_TIMEOUT_ACTION=restart
ReloginAfterTimeoutAction=restart
ExistingSessionDetectedAction=primaryoverride
```

IB server maintenance windows

Period	Time (ET)	Impact
Daily reset	23:45 - 00:45	Expect 1100 → 1101/1102 sequence
Friday extended	23:00 - 03:00	Full maintenance, all services down
Sunday reconnect	~01:00	Weekly 2FA required

Do not schedule critical operations during these windows.

Health check implementation

```python
```

```python
@app.route('/health')
def health_check():
    checks = {
        'ib_connected': ib.isConnected(),
        'last_tick_age_seconds': time.time() - last_tick_timestamp,
        'memory_usage_mb': get_memory_usage(),
        'active_subscriptions': len(ib.tickers())
    }

    healthy = (
        checks['ib_connected'] and
        checks['last_tick_age_seconds'] < 30 and
        checks['memory_usage_mb'] < 1500
    )

    return jsonify({
        'status': 'healthy' if healthy else 'unhealthy',
        'checks': checks
    }), 200 if healthy else 503
```

Data quality validation

```python
def validate_bar(bar, previous_bar=None):
    """Validate incoming OHLCV data"""
    issues = []

    if not (bar.low <= bar.open <= bar.high):
        issues.append("Open outside high/low range")
    if not (bar.low <= bar.close <= bar.high):
        issues.append("Close outside high/low range")
    if bar.high < bar.low:
        issues.append("High less than low")

    if previous_bar:
        change_pct = abs(bar.close - previous_bar.close) / previous_bar.close * 100
        if change_pct > 5:  # 5% threshold for micro futures
            issues.append(f"Possible bad tick: {change_pct:.2f}% change")

    return len(issues) == 0, issues
```

Memory-efficient bar storage

```python
from collections import deque
import numpy as np

class MemoryEfficientBarStore:
    def __init__(self, max_bars=10000):
        self.bars = deque(maxlen=max_bars)  # Auto-eviction

    def add_bar(self, bar):
        # Store as tuple with smaller types
        self.bars.append((
            bar.date.timestamp(),
            np.float32(bar.open),
            np.float32(bar.high),
            np.float32(bar.low),
            np.float32(bar.close),
            np.int32(bar.volume)
        ))
```

Graceful shutdown handler

```python
import signal

def graceful_shutdown(signum, frame):
    logging.info("Shutdown signal received")

    for ticker in ib.tickers():
        ib.cancelMktData(ticker.contract)

    save_state_to_disk()
    ib.disconnect()
    sys.exit(0)

signal.signal(signal.SIGTERM, graceful_shutdown)
signal.signal(signal.SIGINT, graceful_shutdown)
```

Production deployment checklist

Before go-live

- [ ] IB Gateway Docker image pinned to specific stable version
- [ ] Auto-restart configured for 11:45 PM ET
- [ ] CME Real-Time subscription active and API acknowledgement form completed
- [ ] Paper trading data sharing enabled (if using paper for testing)
- [ ] Unique client IDs assigned per application instance
- [ ] Connection state machine handles 1100/1101/1102 correctly
- [ ] Historical data pacing queue implemented
- [ ] Exponential backoff with jitter for retries
- [ ] Health check endpoint implemented
- [ ] Data quality validation on incoming bars
- [ ] Memory-efficient storage with max retention limits
- [ ] Graceful shutdown handlers registered
- [ ] Structured logging with rotation configured
- [ ] CME holiday calendar integrated for market hours detection

Environment variables

```env
# IB Gateway
AUTO_RESTART_TIME=11:45 PM
JAVA_HEAP_SIZE=1024
READ_ONLY_API=yes
TWOFA_TIMEOUT_ACTION=restart
TIME_ZONE=America/Chicago

# Application
DATA_FRESHNESS_THRESHOLD_SECONDS=30
MAX_BARS_IN_MEMORY=10000
PACING_MIN_INTERVAL_SECONDS=15
LOG_LEVEL=INFO
```

Conclusion

The critical "missing pieces" for production IB Gateway futures charting center on **three operational realities**: paper trading requires explicit data sharing configuration (Interactive Brokers) with a 24-hour delay, (Interactive Brokers) historical data always requires subscriptions regardless of delayed mode settings, (GitHub) and

daily auto-restart at 11:45 PM ET is essential for stability. Error handling must distinguish between informational codes (2104, 2106) and actionable codes (1100, 1101), with code **1101 requiring immediate re-subscription** of all market data. For futures rolling, use ContFuture for quick historical analysis but implement custom rolling logic with specific contracts for live charting—roll **8 days before expiration** using calendar-based detection with volume confirmation. These operational patterns cannot be inferred from API documentation alone but are essential for building a resilient production charting system.