JSON Export Guide

Overview

The `export_to_json.py` script exports option chain data from the SQLite database to well-structured JSON files. One JSON file is created per expiration date, with data organized by strike price.

JSON Structure

Each JSON file contains:

Metadata Section

```
{
  "metadata": {
    "fetch_timestamp": "2025-10-05T16:48:23.456789",
    "expiration_date": "2025-10-08",
    "days_to_expiration": 3,
    "underlying_symbol": "SPX",
    "underlying_price": 5923.50,
    "underlying_bid": 5923.25,
    "underlying_ask": 5923.75,
    "total_strikes": 100,
    "strike_range": {
      "min": 5800.0,
      "max": 6000.0
    },
    "export_timestamp": "2025-10-05T17:30:00.123456"
  }
}
```

Metadata Fields:

- `fetch_timestamp`: When the option data was fetched from the market
- `expiration_date`: The expiration date for options in this file
- `days_to_expiration`: Number of days until expiration (at fetch time)
- `underlying_symbol`: The underlying symbol (SPX for SPX/SPXW options)
- `underlying_price`: Mid price of the underlying at fetch time
- `underlying_bid`: Bid price of the underlying at fetch time
- `underlying_ask`: Ask price of the underlying at fetch time
- `total_strikes`: Number of strike prices in this file
- `strike_range`: Min and max strike prices in this file
- `export_timestamp`: When this JSON file was created

Strikes Section

Organized by strike price with CALL and PUT as sub-records:

```
{
  "strikes": [
    {
      "strike_price": 5800.0,
      "call": {
        "symbol": ".SPXW251008C5800",
        "bid_price": 12.5,
        "ask_price": 13.0,
        "mid_price": 12.75,
        "delta": 0.45,
        "gamma": 0.003,
        "theta": -0.25,
        "vega": 0.15,
        "rho": 0.05
      },
      "put": {
        "symbol": ".SPXW251008P5800",
        "bid_price": 8.0,
        "ask_price": 8.5,
        "mid_price": 8.25,
        "delta": -0.35,
        "gamma": 0.003,
        "theta": -0.20,
        "vega": 0.15,
        "rho": -0.04
      }
    }
  ]
}
```


Usage

Basic Export (All Data from Latest Fetch)

```
python export_to_json.py
```

This exports all expirations from the most recent data fetch to `json_exports/` directory.

Export with Filters

Filter by Expiration Date Range

```
# Export options expiring between Oct 6 and Oct 10
python export_to_json.py --start_date 2025-10-06 --end_date 2025-10-10

# Export only options expiring on or after Oct 8
python export_to_json.py --start_date 2025-10-08
```

```
# Export only options expiring on or before Oct 10
python export_to_json.py --end_date 2025-10-10
```

Filter by Strike Range

```
# Export only strikes between 5800 and 6000
python export_to_json.py --min_strike 5800 --max_strike 6000
```

Filter by Delta Range

```
# Export only options with absolute delta between 0.20 and 0.50
python export_to_json.py --min_delta 0.20 --max_delta 0.50
```

Combine Multiple Filters

```
# Export Oct 6-10, strikes 5800-6000, deltas 0.25-0.45
python export_to_json.py \
    --start_date 2025-10-06 \
    --end_date 2025-10-10 \
    --min_strike 5800 \
    --max_strike 6000 \
    --min_delta 0.25 \
    --max_delta 0.45
```

Export Specific Data

Export Specific Expiration Date

```
python export_to_json.py --expiration_date 2025-10-08
```

Export Specific Fetch Timestamp

```
python export_to_json.py --fetch_timestamp "2025-10-05T16:48:23.456789"
```

Custom Output Directory

```
python export_to_json.py --output_dir my_exports
```

Custom Database Path

```
python export_to_json.py --db_path my_data.db --output_dir my_exports
```

CLI Parameters

Parameter	Type	Description	Default
`--db_path`	string	Path to SQLite database file	`database/option_chain_data.db`
`--output_dir`	string	Output directory for JSON files	`json_exports`
`--fetch_timestamp`	string	Specific fetch timestamp (ISO format)	Latest available
`--start_date`	string	Start expiration date (YYYY-MM-DD, inclusive)	No limit
`--end_date`	string	End expiration date (YYYY-MM-DD, inclusive)	No limit
`--min_strike`	float	Minimum strike price	No limit
`--max_strike`	float	Maximum strike price	No limit
`--min_delta`	float	Minimum absolute delta (e.g., 0.20)	No limit
`--max_delta`	float	Maximum absolute delta (e.g., 0.50)	No limit
`--expiration_date`	string	Specific expiration (YYYY-MM-DD)	All expirations

Output Files

File Naming Convention

Files are named using the pattern:

```
spx_options_{expiration_date}_{fetch_datetime}.json
```

Example:

```
spx_options_2025-10-08_20251005_164823.json
```

File Organization

```
json_exports/
├── spx_options_2025-10-06_20251005_164823.json
├── spx_options_2025-10-07_20251005_164823.json
├── spx_options_2025-10-08_20251005_164823.json
└── spx_options_2025-10-09_20251005_164823.json
```

Use Cases

1. Export Near-the-Money Options Only

```
# Delta between 0.30 and 0.70 (ATM-ish)
python export_to_json.py --min_delta 0.30 --max_delta 0.70
```

2. Export This Week's Expirations

```
# Export Oct 6-10 only
python export_to_json.py --start_date 2025-10-06 --end_date 2025-10-10
```

3. Export Specific Strike Range for Analysis

```
# Current SPX around 5900, export ±100 points
python export_to_json.py --min_strike 5800 --max_strike 6000
```

4. Export Deep OTM Options

```
# Very low delta options
python export_to_json.py --max_delta 0.15
```

5. Export Single Expiration for Quick Analysis

```
python export_to_json.py --expiration_date 2025-10-08 --output_dir
quick_export
```

6. Export All Options After a Specific Date

```
# Export everything expiring on or after Oct 15
python export_to_json.py --start_date 2025-10-15
```

Reading JSON Files in Python

```
import json
from pathlib import Path

# Read a single file
with open('json_exports/spx_options_2025-10-08_20251005_164823.json', 'r')
as f:
    data = json.load(f)

# Access metadata
print(f"Expiration: {data['metadata']['expiration_date']}")
print(f"DTE: {data['metadata']['days_to_expiration']}")
print(f"Total strikes: {data['metadata']['total_strikes']}")

# Iterate through strikes
for strike_data in data['strikes']:
    strike = strike_data['strike_price']
    call = strike_data['call']
    put = strike_data['put']

    if call and put:
        print(f"Strike {strike}: Call Delta={call['delta']}, Put Delta=
{put['delta']}")

# Read all files in directory
export_dir = Path('json_exports')
for json_file in sorted(export_dir.glob('spx_options_*.json')):
    with open(json_file, 'r') as f:
        data = json.load(f)
        print(f"Loaded {json_file.name}: {len(data['strikes'])} strikes")
```

Reading JSON Files in JavaScript/Node.js

```
const fs = require('fs');

// Read a single file
const data = JSON.parse(
  fs.readFileSync('json_exports/spx_options_2025-10-
08_20251005_164823.json', 'utf8')
);

// Access data
console.log(`Expiration: ${data.metadata.expiration_date}`);
console.log(`DTE: ${data.metadata.days_to_expiration}`);
```

```
// Iterate through strikes
data.strikes.forEach(strikeData => {
  const { strike_price, call, put } = strikeData;
  if (call && put) {
    console.log(`Strike ${strike_price}: Call ${call.delta}, Put
${put.delta}`);
  }
});
```

Data Analysis Examples

Find ATM Strike

```
import json

with open('json_exports/spx_options_2025-10-08_20251005_164823.json', 'r')
as f:
    data = json.load(f)

# Find strike closest to 0.50 delta
atm_strike = min(
    data['strikes'],
    key=lambda s: abs(s['call']['delta'] - 0.50) if s['call'] else
float('inf')
)
print(f"ATM Strike: {atm_strike['strike_price']}")
```

Calculate Implied Move

```
# ATM straddle price approximates expected move
atm = atm_strike
straddle_price = atm['call']['mid_price'] + atm['put']['mid_price']
print(f"Implied move: ${straddle_price:.2f}")
```

Find All Iron Condor Combinations

```
# Filter by delta for IC legs
for strike_data in data['strikes']:
    call = strike_data['call']
    put = strike_data['put']

    # Short put around -0.30 delta
    if put and -0.35 < put['delta'] < -0.25:
        print(f"Short Put candidate: {strike_data['strike_price']} (Δ=
{put['delta']})")
```

```
    # Short call around 0.30 delta
    if call and 0.25 < call['delta'] < 0.35:
        print(f"Short Call candidate: {strike_data['strike_price']} (Δ=
{call['delta']})")
```

Performance Notes

- Export is fast for typical datasets (< 1 second for 1000+ options)
- JSON files are human-readable and compress well
- Each expiration in a separate file for easier parallel processing
- Filters are applied at database query level for efficiency

Tips

1. **Use filters to reduce file size** - Only export the data you need
2. **Organize by use case** - Use `--output_dir` to separate different exports
3. **Automate exports** - Run after each data fetch to maintain JSON backups
4. **Version control friendly** - JSON format works well with git for tracking changes
5. **Query before export** - Use `query_option_data.py` to verify data first