Metro Train Prediction App - CLI Command Reference

Author: David Morrison

Project Repo: https://github.com/DavMorr/wmata-app

Overview

The Metro Train Prediction App provides a specialized CLI command for managing Metro data synchronization with the WMATA API. This command handles the complex process of fetching, transforming, and storing Metro system data including lines, stations, addresses, and geographic path information.

Command Summary

Command	**Purpose**	**Frequency**
`metro:sync`	Synchronize all Metro data from WMATA API	Daily or as needed

`metro:sync --validate`	Validate cache integrity without sync	Hourly monitoring

Metro Sync Command

Command Signature

bash

```
sail artisan metro:sync {--validate : Validate cache integrity first}
```

Purpose

The `metro:sync` command performs a complete synchronization of Metro system data from the WMATA API, including:

1. **Metro Lines** - All available metro lines (Red, Blue, Green, Orange, Silver, Yellow)
2. **Station Data** - Complete station information including coordinates and line assignments
3. **Station Addresses** - Physical addresses for each station
4. **Station Paths** - Geographic ordering and distance calculations for proper route display

Command Options

Option	Description	Usage
`--validate`	Check cache integrity before performing sync	`metro:sync --validate`

Basic Usage

Standard Sync

bash

```
# Basic synchronization (most common usage)

sail artisan metro:sync
```

Validation Check

bash

```
# Check cache integrity without syncing

sail artisan metro:sync --validate
```

Command Output

Successful Sync

```
Starting Metro data synchronization...
+---------------------+-------+
| Type                | Count |
+---------------------+-------+
| Lines synced        | 6     |
| Stations synced     | 95    |
| Path entries synced | 95    |
+---------------------+-------+
Metro data sync completed successfully!

Stations will now display in proper sequence order
```

Validation Success

bash

```
sail artisan metro:sync --validate
# Output:
Checking cache integrity...

Cache is valid
```

Validation Failure with Sync

bash

```
sail artisan metro:sync --validate
# Output:
Checking cache integrity...
Cache validation failed, proceeding with sync...
Starting Metro data synchronization...

# ... continues with full sync
```

Error Scenarios

API Connection Failure

```
Starting Metro data synchronization...
+---------------------+-------+
| Type                | Count |
+---------------------+-------+
| Lines synced        | 0     |
| Stations synced     | 0     |
| Path entries synced | 0     |
+---------------------+-------+
Errors encountered:
```

- API request failed with status: 500

```
Sync failed: API request failed with status: 500
```

Partial Sync Failure

```
Starting Metro data synchronization...
+---------------------+-------+
| Type                | Count |
+---------------------+-------+
| Lines synced        | 6     |
| Stations synced     | 95    |
| Path entries synced | 0     |
+---------------------+-------+
Errors encountered:
```

- Failed to sync path for line RD: API request failed with status: 401
- Failed to sync path for line BL: Line BL not found

Return Codes

Code	Status	Description
0	SUCCESS	Command completed without errors
1	FAILURE	Command failed due to errors

Command Usage Examples

Development Workflow

Initial Setup

bash

```
# After fresh installation or database reset
cd laravel-app
sail artisan migrate

sail artisan metro:sync
```

Daily Development

bash

```
# Check if sync is needed
sail artisan metro:sync --validate
```

```
# If validation fails, sync data
sail artisan metro:sync
```

Debugging Data Issues

bash

```
# Clear cache and resync
sail artisan cache:clear
sail artisan metro:sync


# Check database after sync
sail artisan tinker
App\Models\Line::count();        // Should be 6
App\Models\Station::count();     // Should be ~95
App\Models\StationPath::count(); // Should be ~95
```

Production Deployment

Initial Production Setup

bash

```
# Production deployment
php artisan migrate --force
php artisan metro:sync


# Verify data
php artisan metro:sync --validate
```

Production Maintenance

bash

```
# Weekly data refresh
php artisan metro:sync


# Log output for monitoring
php artisan metro:sync >> /var/log/metro-sync.log 2>&1
```

Container/Docker Usage

With Laravel Sail

bash

```bash
# Standard Sail usage
sail artisan metro:sync

# Run in background
sail artisan metro:sync &

# Capture output
sail artisan metro:sync | tee sync-output.log
```

Direct Docker Execution

bash

```bash
# If running without Sail
docker exec laravel-app php artisan metro:sync
```

Error Handling

Common Error Types

1. WMATA API Errors

Error: `API request failed with status: 401`

bash

```bash
# Check API key configuration
grep WMATA_API_KEY .env

# Verify API key at https://developer.wmata.com
# Test API key directly
curl -H "api_key: YOUR_KEY" https://api.wmata.com/Rail.svc/json/jLines
```

Error: `API request failed with status: 429`

bash

```bash
# Rate limit exceeded - wait and retry
# Check current rate limit
sail artisan tinker
Cache::get('wmata_api_rate_limit');
```

```
# Clear rate limit if needed
Cache::forget('wmata_api_rate_limit');
```

Error: `API request failed with status: 500`

```
bash
# WMATA API is down - retry later
# Check WMATA status at https://developer.wmata.com
```

2. Database Errors

Error: `SQLSTATE[42S02]: Base table or view not found`

```
bash
# Run migrations first
sail artisan migrate


# Then sync data
sail artisan metro:sync
```

Error: `SQLSTATE[23000]: Integrity constraint violation`

```
bash
# Foreign key constraint issue - reset database
sail artisan migrate:fresh
sail artisan metro:sync
```

3. Configuration Errors

Error: `No application encryption key has been specified`

```
bash
# Check .env file has APP_KEY
grep APP_KEY .env


# Generate key if missing (only if necessary)
sail artisan key:generate
```

Error: `Class 'App\Services\WmataApiService' not found`

```
bash
# Check service provider registration
```

```
cat bootstrap/providers.php
# Ensure WmataServiceProvider is listed


# Clear config cache
sail artisan config:clear
```

Error Recovery Strategies

Graceful Degradation

bash

```
# If sync fails, check what data is available
sail artisan tinker
App\Models\Line::count();
App\Models\Station::count();


# App may work with cached data even if sync fails
```

Manual Data Validation

bash

```
# Validate specific data integrity
sail artisan tinker


# Check for lines without stations
App\Models\Line::whereDoesntHave('stationPaths')->get();


# Check for stations without addresses
App\Models\Station::whereDoesntHave('address')->count();


# Check for orphaned paths
App\Models\StationPath::whereDoesntHave('station')->count();
```

Automation and Scheduling

Laravel Task Scheduler

Basic Scheduling

php

```php
// app/Console/Kernel.php
protected function schedule(Schedule $schedule): void
{
    // Daily sync at 3 AM
    $schedule->command('metro:sync')
             ->dailyAt('03:00')
             ->emailOutputOnFailure('admin@example.com');

    // Hourly validation check
    $schedule->command('metro:sync --validate')
             ->hourly()
             ->skip(function () {
                 // Skip if recent sync was successful
                 return Cache::get('metro_last_sync_success', false);
             });
}
```

Advanced Scheduling with Logging

php

```php
// app/Console/Kernel.php
protected function schedule(Schedule $schedule): void
{
    $schedule->command('metro:sync')
             ->dailyAt('03:00')
             ->appendOutputTo(storage_path('logs/metro-sync.log'))
             ->emailOutputOnFailure('admin@example.com')
             ->before(function () {
                 Log::info('Metro sync starting');
             })
             ->after(function () {
                 Log::info('Metro sync completed');
                 Cache::put('metro_last_sync_success', true, 3600);
             })
             ->onFailure(function () {
                 Log::error('Metro sync failed');
                 Cache::put('metro_last_sync_success', false, 3600);
             });
}
```

Cron Job Setup

Direct Cron Entry

bash

```
# Add to crontab (crontab -e)
# Daily sync at 3 AM
0 3 * * * cd /path/to/laravel-app && php artisan metro:sync >>
/var/log/metro-sync.log 2>&1


# Hourly validation
0 * * * * cd /path/to/laravel-app && php artisan metro:sync --validate >>
/var/log/metro-validation.log 2>&1
```

Docker/Sail Cron Setup

bash

```
# For Sail environment
0 3 * * * cd /path/to/project && ./vendor/bin/sail artisan metro:sync >>
/var/log/metro-sync.log 2>&1
```

Monitoring Automation

Health Check Script

bash

```
#!/bin/bash
# metro-health-check.sh


# Check if recent sync was successful
LAST_SYNC=$(docker exec laravel-app php artisan metro:sync --validate 2>&1)


if [[ $LAST_SYNC == *"Cache is valid"* ]]; then
    echo "Metro sync health: OK"
    exit 0
else
    echo "Metro sync health: FAILED"
    echo "$LAST_SYNC"
    exit 1
fi
```

Monitoring and Logging

Log File Locations

Default Laravel Logs

bash

```
# Application logs
storage/logs/laravel.log


# Custom sync logs (if configured)
storage/logs/metro-sync.log
storage/logs/metro-validation.log
```

Log Monitoring Commands

bash

```
# Real-time log monitoring
tail -f storage/logs/laravel.log


# Search for Metro-specific logs
grep "WMATA" storage/logs/laravel.log
grep "Metro sync" storage/logs/laravel.log


# Check for errors
grep "ERROR" storage/logs/laravel.log | grep -i metro
```

Performance Monitoring

Command Timing

bash

```
# Time command execution
time sail artisan metro:sync


# Example output:
# real    0m45.123s
# user    0m2.456s
# sys     0m0.789s
```

Database Impact Monitoring

bash

```
# Before sync
sail artisan tinker
DB::table('lines')->count();
DB::table('stations')->count();
DB::table('station_paths')->count();


# Run sync with timing
time sail artisan metro:sync


# After sync - verify counts
```

Success Metrics

Key Performance Indicators

- **Sync Duration**: Should complete within 60 seconds
- **Data Counts**: Lines (6), Stations (~95), Paths (~95)
- **Error Rate**: < 5% failure rate over time
- **Cache Hit Rate**: > 90% for prediction requests

Health Check Queries

sql

```
-- Verify data completeness
SELECT
    (SELECT COUNT(*) FROM lines) as line_count,
    (SELECT COUNT(*) FROM stations) as station_count,
    (SELECT COUNT(*) FROM station_paths) as path_count,
    (SELECT COUNT(*) FROM station_addresses) as address_count;


-- Check for recent updates
SELECT
    MAX(updated_at) as last_line_update,
    (SELECT MAX(updated_at) FROM stations) as last_station_update,
    (SELECT MAX(updated_at) FROM station_paths) as last_path_update
FROM lines;
```

Troubleshooting

Debug Mode

Enable Detailed Logging

bash

```
# Temporarily enable debug mode
echo "APP_DEBUG=true" >> .env


# Run sync with verbose output
sail artisan metro:sync -v


# Disable debug mode when done
sed -i 's/APP_DEBUG=true/APP_DEBUG=false/' .env
```

Check Service Registration

bash

```
# Verify services are registered
sail artisan route:list --path=api


# Check service container bindings
sail artisan tinker
app()->bound(App\Services\WmataApiService::class);
app()->bound(App\Services\MetroDataService::class);
```

Data Validation

Manual Data Integrity Checks

bash

```
# Check for missing data
sail artisan tinker


# Lines should have start/end stations
App\Models\Line::whereDoesntHave('startStation')->get();
App\Models\Line::whereDoesntHave('endStation')->get();


# Stations should have addresses
App\Models\Station::whereDoesntHave('address')->count();
```

```
# Paths should be sequential
App\Models\StationPath::where('line_code', 'RD')->orderBy('seq_num')->pluck('seq_num')->toArray();

// Should be [1, 2, 3, 4, 5, ...]
```

Cache Debugging

bash

```
# Check cache contents
sail artisan tinker
Cache::get('wmata.lines');
Cache::get('metro.lines.frontend');
Cache::get('wmata.predictions.A01');


# Clear specific cache keys
Cache::forget('wmata.lines');

Cache::flush(); // Clear all cache
```

Performance Issues

Slow Sync Diagnosis

bash

```
# Check network connectivity to WMATA
curl -w "@curl-format.txt" -o /dev/null -s https://api.wmata.com/Rail.svc/json/jLines


# Monitor database during sync
# In separate terminal:

watch -n 1 "sail artisan tinker --execute='echo App\Models\Station::count();'"
```

Memory Usage Monitoring

bash

```
# Check memory usage during sync
# Monitor with top/htop while running:
sail artisan metro:sync


# Check PHP memory limits
sail artisan tinker

ini_get('memory_limit');
```

Recovery Procedures

Complete Data Reset

bash

```
# Nuclear option - complete reset
sail artisan migrate:fresh

sail artisan metro:sync
```

Selective Data Refresh

bash

```
# Clear only Metro-related cache
sail artisan tinker
Cache::forget('wmata.lines');
Cache::forget('wmata.stations.all');
Cache::forget('metro.lines.frontend');


# Resync without database reset

sail artisan metro:sync
```

This CLI command reference provides comprehensive documentation for managing the Metro Train Prediction App's data synchronization and maintenance operations.