Redis Key → Function Mapping

This section lists each Redis key or namespace and maps all the functions that read, write, or modify it across the system.

News Keys

`news:benzinga:live:raw:{id}.{updated}`

- Written by: `RedisClient.set_news()` (`redisDB/redisClasses.py`, called from `bz_websocket.py`)
- Read by: `BaseProcessor._process_item()` (`redisDB/BaseProcessor.py`, inherited by `NewsProcessor.py`)
- Deleted by: `BaseProcessor._process_item()` if `delete_raw=True`

`news:hist:raw:{id}.{updated}`

- Written by: `RedisClient.set_news_batch()` (`redisDB/redisClasses.py`, called from `bz_restAPI.py`)
- Read by: `BaseProcessor._process_item()` (`redisDB/BaseProcessor.py`, inherited by `NewsProcessor.py`)
- Deleted by: `BaseProcessor._process_item()` if `delete_raw=True`

`news:benzinga:queues:raw`

- Written to: `RedisClient.set_news()` via `LPUSH` (for live news)
- Read by: `BaseProcessor.process_all_items()` via `pop_from_queue()` (`NewsProcessor.py`)

`news:queues:raw`

- Written to: `RedisClient.set_news_batch()` via `LPUSH` (for historical news)
- Read by: `BaseProcessor.process_all_items()` via `pop_from_queue()` (`NewsProcessor.py`)

`news:benzinga:{live|hist}:processed:{id}.{updated}`

- Written by: `BaseProcessor._process_item()` (`redisDB/BaseProcessor.py`, inherited by `NewsProcessor.py`)
- Read by: `ReturnsProcessor._process_single_item()`, `ReturnsProcessor._process_live_news()`, `ReturnsProcessor._process_hist_news()`

`news:benzinga:queues:processed`

- Written to: `BaseProcessor._process_item()` via `LPUSH`
- Used by: `reconcile_missing_items()` for reconciliation and monitoring

news:benzinga:live:processed (channel)

- Published to: `BaseProcessor._process_item()` if live
- Subscribed by: `ReturnsProcessor` (via constructor and `process_all_returns()`)

news:withreturns:{id}

- Written by: `ReturnsProcessor._process_returns()`, `ReturnsProcessor._update_return()`
- Read by: `Neo4jProcessor.process_news_data()`, `reconcile_missing_items()`
- Deleted by: `Neo4jProcessor._handle_ingestion_success()` (neograph/mixins/pubsub.py)

news:withoutreturns:{id}

- Written by: `ReturnsProcessor._process_returns()`
- Read by: `ReturnsProcessor._process_pending_returns()`
- Upgraded to `news:withreturns:{id}` by: `ReturnsProcessor._update_return()`

news:pending_returns (ZSET)

- Written by: `ReturnsProcessor._process_returns()`, `ReturnsProcessor._schedule_pending_returns()`
- Read by: `ReturnsProcessor._process_pending_returns()`, `ReturnsProcessor._sleep_until_next_return()`
- Items removed via `ZREM` by: `ReturnsProcessor._update_return()`

Report Keys

reports:live:raw:{accessionNo}.{filedAt}

- Written by: `RedisClient.set_filing()` (redisDB/redisClasses.py, called from `sec_websocket.py`)
- Read by: `BaseProcessor._process_item()` (redisDB/BaseProcessor.py, inherited by `ReportProcessor.py`)
- Deleted by: `BaseProcessor._process_item()` if `delete_raw=True`

reports:hist:raw:{accessionNo}.{filedAt}

- Written by: `RedisClient.set_filing()` (redisDB/redisClasses.py, called from `sec_restAPI.py`)
- Read by: `BaseProcessor._process_item()` (redisDB/BaseProcessor.py, inherited by `ReportProcessor.py`)
- Deleted by: `BaseProcessor._process_item()` if `delete_raw=True`

reports:queues:raw

- Written to: `RedisClient.set_filing()` via `LPUSH` (both live and hist)
- Read by: `BaseProcessor.process_all_items()` via `pop_from_queue()` (`ReportProcessor.py`)

reports:{live|hist}:processed:{accessionNo}.{filedAt}

- Written by: `BaseProcessor._process_item()` (redisDB/BaseProcessor.py, inherited by `ReportProcessor.py`)
- Read by: `ReturnsProcessor._process_single_item()`, `ReturnsProcessor._process_live_news()`, `ReturnsProcessor._process_hist_news()`

reports:queues:processed

- Written to: `BaseProcessor._process_item()` via `LPUSH`
- Used for batch ingestion or reconciliation

reports:live:processed (channel)

- Published to: `BaseProcessor._process_item()` if live
- Subscribed by: `ReturnsProcessor` (via constructor and `process_all_returns()`)

reports:withreturns:{id}

- Written by: `ReturnsProcessor._process_returns()`, `ReturnsProcessor._update_return()`
- Read by: `Neo4jProcessor.process_report_data()`, `reconcile_missing_items()`
- Deleted by: `Neo4jProcessor._handle_ingestion_success()` (neograph/mixins/pubsub.py)

reports:withoutreturns:{id}

- Written by: `ReturnsProcessor._process_returns()`
- Read by: `ReturnsProcessor._process_pending_returns()`
- Upgraded to `reports:withreturns:{id}` by: `ReturnsProcessor._update_return()`

reports:pending_returns (ZSET)

- Written by: `ReturnsProcessor._process_returns()`, `ReturnsProcessor._schedule_pending_returns()`
- Read by: `ReturnsProcessor._process_pending_returns()`, `ReturnsProcessor._sleep_until_next_return()`
- Items removed via `ZREM` by: `ReturnsProcessor._update_return()`

Transcript Keys

transcripts:{live|hist}:raw:{symbol}_{timestamp}

- Written by: `EarningsCallProcessor.store_transcript_in_redis()` (transcripts/Earnin
- Read by: `BaseProcessor._process_item()` (redisDB/TranscriptProcessor.py)
- Deleted by: `TranscriptProcessor._handle_transcript_found()`

transcripts:queues:raw

- Written to: `EarningsCallProcessor.store_transcript_in_redis()` via `LPUSH`

- Read by: `BaseProcessor.process_all_items()` via `pop_from_queue()` (`TranscriptProcessor.py`)

`admin:transcripts:schedule` (ZSET)

- Written to: `EarningsCallProcessor.store_transcript_in_redis()` (adds items with timestamp scores)
- Written to: `TranscriptProcessor._schedule_transcript_retry()` (reschedules with new timestamp)
- Read by: `TranscriptProcessor._process_due_transcripts()` via `ZRANGE`
- Items removed via `ZREM` by: `TranscriptProcessor._handle_transcript_found()`

`admin:transcripts:notifications` (channel)

- Published to: `EarningsCallProcessor.store_transcript_in_redis()` (notifies about new transcript)
- Published to: `TranscriptProcessor._handle_transcript_found()` (success notification)
- Published to: `TranscriptProcessor._schedule_transcript_retry()` (retry notification)
- Subscribed by: `TranscriptProcessor._run_transcript_scheduling()`

`transcripts:{live|hist}:processed:{symbol}_{timestamp}`

- Written by: `BaseProcessor._process_item()` (`redisDB/TranscriptProcessor.py`)
- Read by: `Neo4jProcessor.process_transcript_data()`

`transcripts:queues:processed`

- Written to: `BaseProcessor._process_item()` via `LPUSH`
- Used for reconciliation

`transcripts:withreturns:{id}`

- Written by: `ReturnsProcessor._process_returns()` (for transcripts)
- Read by: `Neo4jProcessor.process_transcript_data()`, `reconcile_missing_items()`
- Deleted by: `Neo4jProcessor._handle_ingestion_success()` (`neograph/mixins/pubsub.py`)

`transcripts:withoutreturns:{id}`

- Written by: `ReturnsProcessor._process_returns()` (for transcripts)
- Upgraded via same mechanism as other source types

`admin:transcripts:processed` (SET)

- Written to: `TranscriptProcessor._handle_transcript_found()` via `SADD`
- Read by: `TranscriptProcessor._process_due_transcripts()` for deduplication

Admin & Configuration Keys

`batch:{source}:{from}-{to}:fetch_complete`

- Written by: `set_news_batch()`, `set_filing()` (for batch operations)
- Read by: `DataManagerCentral`, `gap_fill_runner.py`, `run_event_trader.py`

`admin:operations:{type}:{id}`

- Written/incremented by: `StatsTracker.increment()`, `StatsTracker.set_status()`
- Read by: `StatsTracker.get_stats()` for monitoring dashboards

`admin:tradable_universe:symbols` **(SET)**

- Written by: `EventTraderRedis.initialize_stock_universe()`
- Read by: `RedisClient.get_symbols()`, `BaseProcessor.__init__()` for symbol validation

`admin:tradable_universe:stock_universe`

- Written by: `EventTraderRedis.initialize_stock_universe()`
- Read by: `RedisClient.get_stock_universe()` for ETF/sector information

`admin:{news|reports}:shutdown_state`

- Written by: `disconnect()` in WebSocket classes to record clean shutdown
- Read by: `connect()` in WebSocket classes to detect restart gaps

`admin:{news|reports}:last_message_time`

- Written by: WebSocket `_on_message()` handlers
- Read by: WebSocket classes on initialization

`admin:backfill:{news|reports}_restart_gap`

- Written by: `connect()` in WebSocket classes when restart detected
- Read by: Gap-filling scripts for data reconciliation

`admin:websocket_downtime:{source}:{timestamp}`

- Written by: `_log_downtime()` in WebSocket classes
- Used for operational monitoring and gap identification
