# Product Attribute Mapping Analytics - Engineering Implementation Plan

## Problem Statement

We need to track and send analytics data to Meta about:

1. **Products with mapped attributes**: Count of products that have at least one mapped attribute with a value
2. **Total attribute mappings**: Total count of all mapped attributes across all products

This data should be sent to Meta automatically as background telemetry to understand mapping usage patterns.

## Requirements

### Functional Requirements

* Track number of products using mapped attributes
* Track total count of attribute mappings across all products
* Send analytics data to Meta automatically
* No UI required - pure background functionality
* Minimal performance impact on existing operations

### Non-Functional Requirements

* Run only when necessary (not on every product edit)
* Use existing plugin infrastructure
* Maintain data accuracy
* Handle large product catalogs efficiently

## Solution Analysis

### Option 1: Batch API Integration ❌

**Approach**: Hook into individual product sync operations via batch API.

**Pros**:

* Real-time tracking
* Incremental updates

**Cons**:

* Only captures products being actively edited
* Incomplete data (doesn't cover all products)
* Would require complex accumulation logic
* Performance impact on every product edit

**Verdict**: Rejected - doesn't provide complete store-wide analytics

### Option 2: Scheduled Background Job ⚠️

**Approach**: Daily/weekly scheduled job to scan all products.

**Pros**:

* Complete data coverage
* Predictable timing
* No impact on user operations

**Cons**:

* Additional scheduling complexity
* Duplicate logic with existing product processing
* Potential performance issues with large catalogs

**Verdict**: Viable but not optimal

### Option 3: Feed Generation Hook ✅ (Chosen)

**Approach**: Hook into existing feed generation process.

**Pros**:

* Complete data coverage (all products)
* Leverages existing product processing
* Efficient timing (when feed is generated)
* Uses existing infrastructure
* No additional scheduling needed

**Cons**:

* Tied to feed generation frequency
* Slightly delays feed completion

**Verdict**: Optimal solution

## Implementation Plan

### Phase 1: Core Analytics Calculation

| **class** ProductAttributeAnalytics {  */\*\**  *\* Calculate store-wide attribute mapping statistics*  *\*/*  **public** **static** **function** calculate\_mapping\_analytics() {  $products\_with\_mappings = 0;  $total\_attribute\_mappings = 0;    *// Use same product query as feed generation*  $product\_ids = WC\_Facebookcommerce\_Utils::get\_all\_product\_ids\_for\_sync();    **foreach** (product\_ids **as** product\_id) {  product = wc\_get\_product(product\_id);  **if** (!$product) **continue**;    *// Get mapped attributes for this product*  mapped\_attributes = ProductAttributeMapper::get\_mapped\_attributes(product);    **if** (!**empty**($mapped\_attributes)) {  $products\_with\_mappings++;  total\_attribute\_mappings += count(mapped\_attributes);  }  }    **return** [  'products\_with\_mapped\_attributes' => $products\_with\_mappings,  'total\_attribute\_mappings' => $total\_attribute\_mappings,  'total\_products\_processed' => count($product\_ids),  'timestamp' => current\_time('mysql')  ];  }  } |
| --- |

### Phase 2: Integration with Feed Generation

| *// Hook into feed generation completion*  add\_action('wc\_facebook\_feed\_generation\_completed', 'send\_attribute\_analytics\_to\_meta');  **function** send\_attribute\_analytics\_to\_meta() {  *// Optional: Throttle to max once per day*  $last\_sent = get\_option('wc\_facebook\_last\_analytics\_sent', 0);  **if** (time() - $last\_sent < DAY\_IN\_SECONDS) {  **return**;  }    *// Calculate analytics*  $analytics = ProductAttributeAnalytics::calculate\_mapping\_analytics();    *// Send via existing logging system*  Logger::log(  'Product attribute mapping analytics',  $analytics,  **array**(  'should\_send\_log\_to\_meta' => **true**,  'should\_save\_log\_in\_woocommerce' => **true**,  'woocommerce\_log\_level' => \WC\_Log\_Levels::DEBUG,  )  );    *// Update last sent timestamp*  update\_option('wc\_facebook\_last\_analytics\_sent', time());  } |
| --- |

### Phase 3: Error Handling & Optimization

| **function** send\_attribute\_analytics\_to\_meta() {  **try** {  $analytics = ProductAttributeAnalytics::calculate\_mapping\_analytics();    Logger::log(  'Product attribute mapping analytics',  $analytics,  **array**(  'should\_send\_log\_to\_meta' => **true**,  'should\_save\_log\_in\_woocommerce' => **true**,  'woocommerce\_log\_level' => \WC\_Log\_Levels::DEBUG,  )  );    update\_option('wc\_facebook\_last\_analytics\_sent', time());    } **catch** (Exception $e) {  Logger::log(  'Failed to send attribute analytics: ' . $e->getMessage(),  [],  **array**(  'should\_send\_log\_to\_meta' => **false**,  'should\_save\_log\_in\_woocommerce' => **true**,  'woocommerce\_log\_level' => \WC\_Log\_Levels::ERROR,  )  );  }  } |
| --- |

## Technical Tradeoffs

### Data Transmission Method

| Option | Pros | Cons | Decision |
| --- | --- | --- | --- |
| Batch API | Product-focused | Wrong use case | ❌ |
| Custom API | Full control | Additional complexity | ❌ |
| Existing Logger | Proven, integrated | Limited to log format | ✅ |

**Chosen**: Existing Logger system

* Already handles Meta authentication
* Consistent with plugin patterns
* Proven reliability

### Timing Strategy

| Option | Pros | Cons | Decision |
| --- | --- | --- | --- |
| Real-time | Immediate | Incomplete data | ❌ |
| Scheduled | Predictable | Additional complexity | ❌ |
| Feed Hook | Complete data | Tied to feed timing | ✅ |

**Chosen**: Feed Generation Hook

* Guarantees complete data coverage
* Efficient resource usage
* No additional scheduling overhead

### Performance Considerations

| Aspect | Impact | Mitigation |
| --- | --- | --- |
| Product Query | Moderate | Reuse existing feed query logic |
| Attribute Calculation | Low | Leverage existing get\_mapped\_attributes() |
| Timing | Minimal | Runs only during feed generation |

## Implementation Timeline

### Week 1: Core Development

* [ ] Create ProductAttributeAnalytics class
* [ ] Implement calculate\_mapping\_analytics() method
* [ ] Add basic error handling

### Week 2: Integration

* [ ] Hook into feed generation completion
* [ ] Implement throttling mechanism
* [ ] Add logging integration

### Week 3: Testing & Optimization

* [ ] Test with large product catalogs
* [ ] Verify data accuracy
* [ ] Performance testing

## Monitoring & Validation

### Success Metrics

* Analytics data successfully sent to Meta
* Minimal performance impact on feed generation
* Data accuracy verified against manual counts

### Failure Scenarios

* Feed generation disabled → No analytics sent (acceptable)
* Product query failure → Logged error, no analytics sent
* Logger failure → Logged locally, retry on next feed generation

## Future Considerations

### Potential Enhancements

* Historical trend tracking
* More granular attribute usage statistics
* Integration with other sync methods (if needed)

### Maintenance

* Monitor for changes in feed generation process
* Update if ProductAttributeMapper API changes
* Adjust throttling based on usage patterns

## Dependencies

### Existing Systems

* ProductAttributeMapper::get\_mapped\_attributes()
* WC\_Facebookcommerce\_Utils::get\_all\_product\_ids\_for\_sync()
* Plugin Logger system
* Feed generation process

### WordPress/WooCommerce

* WordPress Options API (get\_option, update\_option)
* WooCommerce product system
* Action hooks (wc\_facebook\_feed\_generation\_completed)

## Conclusion

The chosen approach provides complete, accurate analytics data with minimal system impact by leveraging existing infrastructure. The feed generation hook ensures we capture all products in the store while maintaining efficiency through strategic timing and throttling.