Function 5: Batch Fetch Records to JSON

Overview

Function 5 performs bulk extraction of bibliographic record data from Alma, saving the complete XML of multiple records to a single JSON file. This function is essential for large-scale data analysis, backup, migration, or external processing of Alma records.

What It Does

This function retrieves the full XML content of all records in a loaded set and exports them to a structured JSON file with:

- Each record's MMS ID as the key
- Complete XML as the value
- Proper JSON formatting and escaping
- UTF-8 character encoding
- Timestamped filename for versioning

Key Features

- **Batch processing**: Handles any number of records in a set
- **Complete records**: Exports full bibliographic XML
- **JSON format**: Machine-readable, parseable output
- **Progress tracking**: Real-time progress bar during export
- **Kill switch**: Can stop export mid-process
- **Error handling**: Continues on individual record failures
- **Automatic file naming**: Uses timestamp and record count
- **UTF-8 support**: Preserves special characters

The Need for This Function

Data Export and Backup

Alma's standard export tools have limitations:

- Limited to specific fields or formats
- May not include complete XML
- Not optimized for programmatic access
- Difficult to process large sets

Function 5 provides:

- Complete XML export
- Programmatically accessible format (JSON)
- Easy parsing with standard libraries
- Full control over exported data

Analysis and Processing

Common scenarios requiring bulk XML access:

- **Data quality analysis**: Examine metadata patterns across records
- **Migration preparation**: Extract data for transformation
- **External processing**: Feed to XSLT, Python scripts, or other tools
- **Backup before changes**: Snapshot state before bulk edits
- **API development**: Test data for applications
- **Documentation**: Examples for training or specifications

How It Works

Step-by-Step Process

1. **Load Set**:
    - User enters set ID
    - Clicks "Load Set"
    - Set members retrieved from Alma
    - MMS IDs stored in application state
2. **Select Function**:
    - Choose "Batch Fetch Records to JSON" from dropdown
    - Function 5 button appears
3. **Execute**:
    - Click function button
    - Progress bar appears
    - For each MMS ID:
        - Send GET request to Alma Bibs API
        - Receive full XML response
        - Store in dictionary: `{mms_id: xml_string}`
4. **Progress Tracking**:
    - Progress bar updates after each record
    - Shows "Processing record X of Y"
    - Percentage completion visible
    - Can click "Kill" to stop
5. **JSON Generation**:
    - Convert dictionary to JSON
    - Pretty-print with indentation
    - Ensure UTF-8 encoding
    - Escape special characters
6. **File Output**:

  - Generate filename: `batch_records_YYYYMMDD_HHMMSS_NNN.json`
  - Write JSON to file in project directory
  - Log file location
  - Display success message

Data Structure

Python Dictionary (internal):

```
{
    "9912345678901O4641": "<bib>...</bib>",
    "9912345678902O4641": "<bib>...</bib>",
    "9912345678903O4641": "<bib>...</bib>",
    ...
}
```

JSON Output File:

```
{
  "991234567890104641": "<?xml version=\"1.0\" ?>\n<bib>\n
<mms_id>991234567890104641</mms_id>\n
<record_format>marc21</record_format>\n  ...\n</bib>",
  "991234567890204641": "<?xml version=\"1.0\" ?>\n<bib>\n
<mms_id>991234567890204641</mms_id>\n
<record_format>marc21</record_format>\n  ...\n</bib>",
  "991234567890304641": "<?xml version=\"1.0\" ?>\n<bib>\n
<mms_id>991234567890304641</mms_id>\n
<record_format>marc21</record_format>\n  ...\n</bib>"
}
```

Usage

Basic Export

Step 1: Load Set

1. Enter set ID in "Set ID" field
   - Example: `7071087320004641` (DCAP01 set)
   - Or click DCAP01 link to auto-populate
2. Click "Load Set" button
3. Wait for confirmation: "Set loaded: 2,847 records"

Step 2: Select Function

1. Open function dropdown
2. Select "Batch Fetch Records to JSON"
3. Function 5 button becomes active

Step 3: Execute Export

1. Click function 5 button
2. Progress bar appears
3. Watch progress: "Processing record 1 of 2,847"
4. Wait for completion (may take 1-2 hours for large sets)

Step 4: Locate Output File

1. Check CABB project directory
2. Find file: `batch_records_20241203_143022_2847.json`
3. File contains JSON with all record XML

Step 5: Verify Export

1. Open JSON file in text editor
2. Verify record count matches set size
3. Check sample records for completeness
4. Confirm UTF-8 encoding preserved

Kill Switch Usage

When to Use:

- Export taking too long
- Need to stop for system maintenance
- Discovered wrong set was loaded
- Want partial export for testing

How to Use:

1. During export, progress bar shows "Kill" button
2. Click "Kill" button
3. Current record completes
4. Export stops
5. Partial results saved to JSON file
6. Filename reflects actual record count exported

Example:

- Set has 2,847 records
- Kill after 500 records
- Output file: `batch_records_20241203_143022_500.json`
- Contains first 500 records only

Large Set Handling

For sets with 1,000+ records:

Strategy 1: Full Export Overnight

- Start export at end of day

- Let run overnight
- Review results in morning
- Typical: 2,847 records = ~2 hours

Strategy 2: Subset Exports

- Create smaller temporary sets in Alma
- Export each subset separately
- Combine JSON files afterward
- More control, easier to restart

Strategy 3: Progressive Exports

- Export first 500 records
- Use kill switch
- Verify partial export
- Resume with new set (remaining records)

Output File Format

Filename Convention

Pattern: `batch_records_YYYYMMDD_HHMMSS_COUNT.json`

Components:

- `batch_records`: Fixed prefix
- `YYYYMMDD`: Date (e.g., 20241203 = December 3, 2024)
- `HHMMSS`: Time (e.g., 143022 = 2:30:22 PM)
- `COUNT`: Number of records exported
- `.json`: File extension

Examples:

- `batch_records_20241203_143022_2847.json` - Full DCAP01 export
- `batch_records_20241203_150000_500.json` - Killed after 500 records
- `batch_records_20241203_160000_1.json` - Single record test

JSON Structure

Root Object:

```
{
  "MMS_ID_1": "XML_STRING_1",
  "MMS_ID_2": "XML_STRING_2",
  ...
}
```

Key: MMS ID (string)

- 21-digit Alma record identifier
- Example: "991234567890104641"

Value: XML (string)

- Complete bibliographic record XML
- Escaped for JSON (quotes, newlines, etc.)
- Includes XML declaration
- UTF-8 encoded

Pretty Printing:

- Indent: 2 spaces
- Ensure ASCII: False (allows Unicode)
- Sort keys: False (maintains insertion order)

Character Encoding

UTF-8 Throughout:

- File written with UTF-8 encoding
- Special characters preserved
- JSON escape sequences for quotes/newlines
- No mojibake or data corruption

Example Special Characters:

- Accented letters: é, ñ, ü
- Quotes: "", '', ""
- Dashes: —, –
- Symbols: ©, ®, °

Use Cases

1. Pre-Edit Backup

Scenario: About to run Functions 2, 6, or 7 on large set, want backup first

Workflow:

1. Load set (e.g., DCAP01 with 2,847 records)
2. Run Function 5 to export all records
3. Store JSON file safely
4. Run editing functions (2, 6, 7)
5. If issues arise, have complete pre-edit state

Benefits:

- Complete snapshot of data before changes
- Can analyze what changed
- Recovery option if needed
- Documentation of original state

2. Data Quality Analysis

Scenario: Analyze metadata patterns across collection

Workflow:

1. Export entire collection to JSON
2. Write Python script to parse JSON:

```python
import json
import xml.etree.ElementTree as ET

with open('batch_records_20241203_143022_2847.json', 'r',
encoding='utf-8') as f:
    records = json.load(f)

# Analyze each record
for mms_id, xml_string in records.items():
    root = ET.fromstring(xml_string)
    # Extract and analyze specific fields
```

3. Generate reports on:
   - Missing fields
   - Field value patterns
   - Data quality issues
   - Metadata completeness

Benefits:

- Comprehensive analysis
- Identify systematic issues
- Document metadata quality
- Inform cleanup priorities

3. Migration or Transformation

Scenario: Migrate metadata to different system or format

Workflow:

1. Export all records from Alma
2. Process JSON with transformation scripts:
   - Extract Dublin Core
   - Convert to MODS, EAD, or other format
   - Map to new system's schema
3. Import to target system
4. Verify migration completeness

Benefits:

- Single export contains all data
- Process offline
- Repeatable transformation
- Version control for scripts

4. External Application Development

Scenario: Building web application that displays Alma records

Workflow:

1. Export test data set
2. Use JSON for development/testing
3. Parse XML to extract display fields
4. Test application against real data
5. Deploy with live API integration

Benefits:

- Realistic test data
- No API calls during development
- Fast iteration
- Consistent test dataset

5. Metadata Auditing

Scenario: Document metadata state for annual report or compliance

Workflow:

1. Export collection at end of fiscal year
2. Archive JSON file with date
3. Generate statistics from JSON:
   - Total records
   - Field usage frequencies
   - Rights statements distribution
   - Format types breakdown
4. Include in annual report

Benefits:

- Point-in-time snapshot
- Reproducible statistics
- Compliance documentation
- Year-over-year comparison

6. Training and Documentation

Scenario: Create training materials showing real record examples

Workflow:

1. Export sample set of diverse records
2. Extract specific examples:
    - Best practice records
    - Problematic records
    - Various content types
3. Use in training documentation
4. Provide to staff for reference

Benefits:

- Real examples from production
- Diverse record types
- Easily shareable
- Version controlled

Technical Details

API Endpoint

For Each Record:

```
GET /almaws/v1/bibs/{mms_id}?view=full&expand=None
Accept: application/xml
Authorization: apikey {api_key}
```

Parameters:

- `mms_id`: Bibliographic record identifier
- `view=full`: Returns complete record data
- `expand=None`: No additional linked data
- API key from environment variables

Response:

- Content-Type: application/xml
- Body: Complete `<bib>...</bib>` XML
- Status: 200 on success

JSON Generation

Python Code Pattern:

```python
import json

# Dictionary to store records
records_dict = {}

# Fetch each record
for mms_id in set_members:
    xml_string = fetch_record_xml(mms_id)
    records_dict[mms_id] = xml_string

# Write to JSON file
filename = f"batch_records_{timestamp}_{len(records_dict)}.json"
with open(filename, 'w', encoding='utf-8') as f:
    json.dump(records_dict, f, indent=2, ensure_ascii=False)
```

JSON Settings:

- `indent=2`: Pretty-print with 2-space indentation
- `ensure_ascii=False`: Allow Unicode characters
- `encoding='utf-8'`: UTF-8 file encoding
- Default key order: insertion order (Python 3.7+)

File I/O

Write Mode: `'w'` (write, overwrite if exists)

Encoding: `'utf-8'` (explicit UTF-8)

Location: CABB project directory (`/Users/mcfatem/GitHub/CABB/`)

Permissions: Uses default system permissions

Performance Considerations

Time per Record:

- API call: 1-2 seconds
- JSON processing: negligible
- Total: ~1.5 seconds average per record

Total Time Estimates:

- 100 records: 2-3 minutes
- 500 records: 12-15 minutes
- 1,000 records: 25-30 minutes
- 2,847 records: 1-2 hours

File Sizes:

- Average record XML: ~15-20 KB
- 100 records: ~1.5-2 MB JSON
- 1,000 records: ~15-20 MB JSON
- 2,847 records: ~43-57 MB JSON

Factors Affecting Speed:

- Network latency
- Alma server load
- Record complexity (large records take longer)

- Time of day (peak vs. off-peak)

Error Handling

Individual Record Failures:

- Error logged with MMS ID and status code
- Record skipped in output JSON
- Processing continues to next record
- Final count reflects successful exports only

Common Errors:

Error	Status	Cause	Handling
Not found	404	Invalid MMS ID	Skip, log error
Unauthorized	401	API key expired	Stop, display error
Forbidden	403	Insufficient permissions	Stop, display error
Timeout	-	Network issue	Skip, log error, continue
Rate limit	429	Too many requests	Retry with delay

Network Failures:

- Logged with full traceback
- User notified of issue
- Can retry entire export or use kill switch

Parsing the JSON Output

Python Example

Load JSON:

```
import json
import xml.etree.ElementTree as ET

# Read JSON file
with open('batch_records_20241203_143022_2847.json', 'r', encoding='utf-8')
as f:
    records = json.load(f)

print(f"Loaded {len(records)} records")

# Access specific record
mms_id = "991234567890104641"
xml_string = records[mms_id]

# Parse XML
root = ET.fromstring(xml_string)
```

```python
# Extract fields
title = root.find('.//title').text
print(f"Title: {title}")
```

Extract Dublin Core:

```python
namespaces = {
    'dc': 'http://purl.org/dc/elements/1.1/',
    'dcterms': 'http://purl.org/dc/terms/'
}

# Find Dublin Core section
record_elem = root.find('.//record[@xmlns]')

# Get all dc:title elements
titles = record_elem.findall('.//dc:title', namespaces)
for title in titles:
    print(f"Title: {title.text}")

# Get all dc:creator elements
creators = record_elem.findall('.//dc:creator', namespaces)
for creator in creators:
    print(f"Creator: {creator.text}")
```

Iterate All Records:

```python
for mms_id, xml_string in records.items():
    try:
        root = ET.fromstring(xml_string)
        # Process record
        process_record(root, mms_id)
    except ET.ParseError as e:
        print(f"Error parsing {mms_id}: {e}")
```

JavaScript Example

Load JSON in Node.js:

```javascript
const fs = require('fs');
const { DOMParser } = require('xmldom');

// Read JSON file
const data = fs.readFileSync('batch_records_20241203_143022_2847.json',
'utf-8');
const records = JSON.parse(data);
```

```
console.log(`Loaded ${Object.keys(records).length} records`);

// Parse specific record
const mmsId = '991234567890104641';
const xmlString = records[mmsId];

const parser = new DOMParser();
const xmlDoc = parser.parseFromString(xmlString, 'text/xml');

// Extract title
const titleElement = xmlDoc.getElementsByTagName('title')[0];
const title = titleElement.textContent;
console.log(`Title: ${title}`);
```

Browser Example:

```
// Assuming JSON loaded as 'records' object

// Iterate all records
Object.entries(records).forEach(([mmsId, xmlString]) => {
  const parser = new DOMParser();
  const xmlDoc = parser.parseFromString(xmlString, 'text/xml');

  // Extract metadata
  const title = xmlDoc.querySelector('title')?.textContent;
  console.log(`${mmsId}: ${title}`);
});
```

Command Line (jq)

Count Records:

```
jq 'length' batch_records_20241203_143022_2847.json
```

List All MMS IDs:

```
jq 'keys[]' batch_records_20241203_143022_2847.json
```

Extract Specific Record:

```
jq '.["991234567890104641"]' batch_records_20241203_143022_2847.json
```

Pretty Print Specific XML:

```
jq -r '.["991234567890104641"]' batch_records_20241203_143022_2847.json |
xmllint --format -
```

Best Practices

Before Export

1. **Verify set membership**: Check set contains intended records
2. **Estimate time**: Calculate expected duration based on record count
3. **Check disk space**: Ensure sufficient space for output file
4. **Test with small set**: Export 10-20 records first to verify
5. **Note timestamp**: Document when export starts for file identification

During Export

1. **Monitor progress**: Check progress bar periodically
2. **Don't close application**: Keep browser window open
3. **Avoid system sleep**: Disable sleep mode for long exports
4. **Check logs**: Review log file if errors appear
5. **Use kill switch wisely**: Only stop if necessary

After Export

1. **Verify file created**: Check project directory for JSON file
2. **Validate JSON**: Use JSON validator to ensure well-formed
3. **Check record count**: Compare file count to set count
4. **Sample records**: Parse and examine a few records
5. **Backup file**: Copy to secure location if important
6. **Document export**: Note date, purpose, and set details

File Management

1. **Descriptive naming**: Include date, set name in filename if renaming
2. **Version control**: Keep exports in dated folders
3. **Compression**: Gzip large files for storage (can compress to ~10% of size)
4. **Retention policy**: Delete old exports after specific period
5. **Security**: Protect files if they contain sensitive metadata

Limitations

- **Set-based only**: Cannot export arbitrary MMS ID list (must be in set)
- **No filtering**: Exports all records in set, no field-level filtering
- **JSON only**: Does not support other formats (CSV, XML file per record, etc.)
- **Full records**: Cannot export subset of fields (always complete XML)
- **No compression**: Output file not automatically compressed
- **Single file**: All records in one JSON file (can be large)
- **No resume**: If export fails, must restart from beginning

- **Memory usage**: Large sets may require significant memory

Troubleshooting

Export Hangs or Stalls

Symptoms: Progress bar stops updating

Possible Causes:

- Network interruption
- Alma server timeout
- Very large record taking long time

Solutions:

- Wait 2-3 minutes before using kill switch
- Check network connection
- Review logs for error messages
- Use kill switch and retry

JSON File Corrupted

Symptoms: Cannot parse JSON, syntax errors

Possible Causes:

- Export interrupted
- File system error
- Character encoding issue

Solutions:

- Use JSON validator to identify problem
- Check if file ends abruptly (missing closing brace)
- Re-export if severely corrupted
- Contact support if persistent

File Not Found After Export

Symptoms: Export completes but file not in directory

Possible Causes:

- Saved to different directory
- Permissions issue
- Filename different than expected

Solutions:

- Search entire system for "batch_records*.json"
- Check user has write permissions to CABB directory

- Review logs for actual filename
- Check for error messages during save

Special Characters Display Incorrectly

Symptoms: Accents, symbols appear as �� or ?

Possible Causes:

- File not opened with UTF-8 encoding
- Editor doesn't support UTF-8
- Character encoding lost

Solutions:

- Open file with UTF-8 encoding explicitly
- Use editor with good Unicode support (VS Code, Sublime)
- Verify JSON file itself is UTF-8 (check with file command)
- Re-export if file truly corrupted

Partial Records in JSON

Symptoms: Some MMS IDs missing from output

Possible Causes:

- Records returned 404 (deleted or invalid)
- API errors for specific records
- Kill switch used

Solutions:

- Check error log for failed MMS IDs
- Verify those records exist in Alma
- Compare file count to expected count
- Re-export missing records individually if needed

Integration with Other Functions

Before Function 2, 6, or 7 (Editing Functions)

Backup Workflow:

1. Load set to be edited
2. Run Function 5 to export all records
3. Verify export completed successfully
4. Store JSON file securely
5. Run editing function
6. Compare results using JSON backup

With Function 3 (CSV Export)

Complementary Use:

- Function 3: Tabular data for spreadsheet analysis
- Function 5: Complete XML for programmatic processing
- Export both formats for different purposes
- CSV for human review, JSON for scripts

With Function 1 (Single XML View)

Detailed Inspection:

- Use Function 5 for bulk export
- Use Function 1 to examine individual records
- Cross-reference between file and live data
- Verify specific records after export

Related Documentation

- **Alma Bibs API**: https://developers.exlibrisgroup.com/alma/apis/bibs/
- **JSON Format**: https://www.json.org/
- **Python json module**: https://docs.python.org/3/library/json.html
- **XML Processing**: https://docs.python.org/3/library/xml.etree.elementtree.html

Version History

- **Initial Implementation**: Batch export capability
- **Purpose**: Bulk data extraction for analysis and backup
- **Status**: Active, production-ready