SmartBulk Technical Guide v2

CloudWithAbhi
Business Edition PDF

SmartBulk - Technical Guide

Package: SmartBulk

Version: 2.3.0

Package ID: 0HoT1000000001dKAA

Version ID: 04tT1000000EMVJIA4

Namespace: smartbulk

Publisher: CloudWithAbhi

Document Version: 1.0

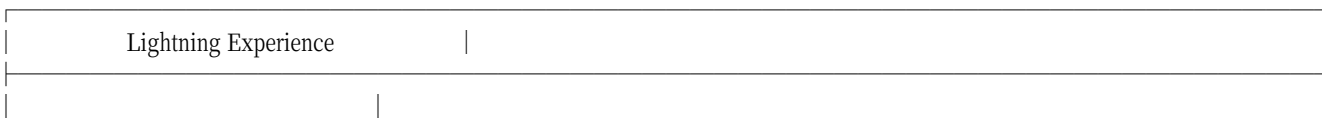
Last Updated: December 2025

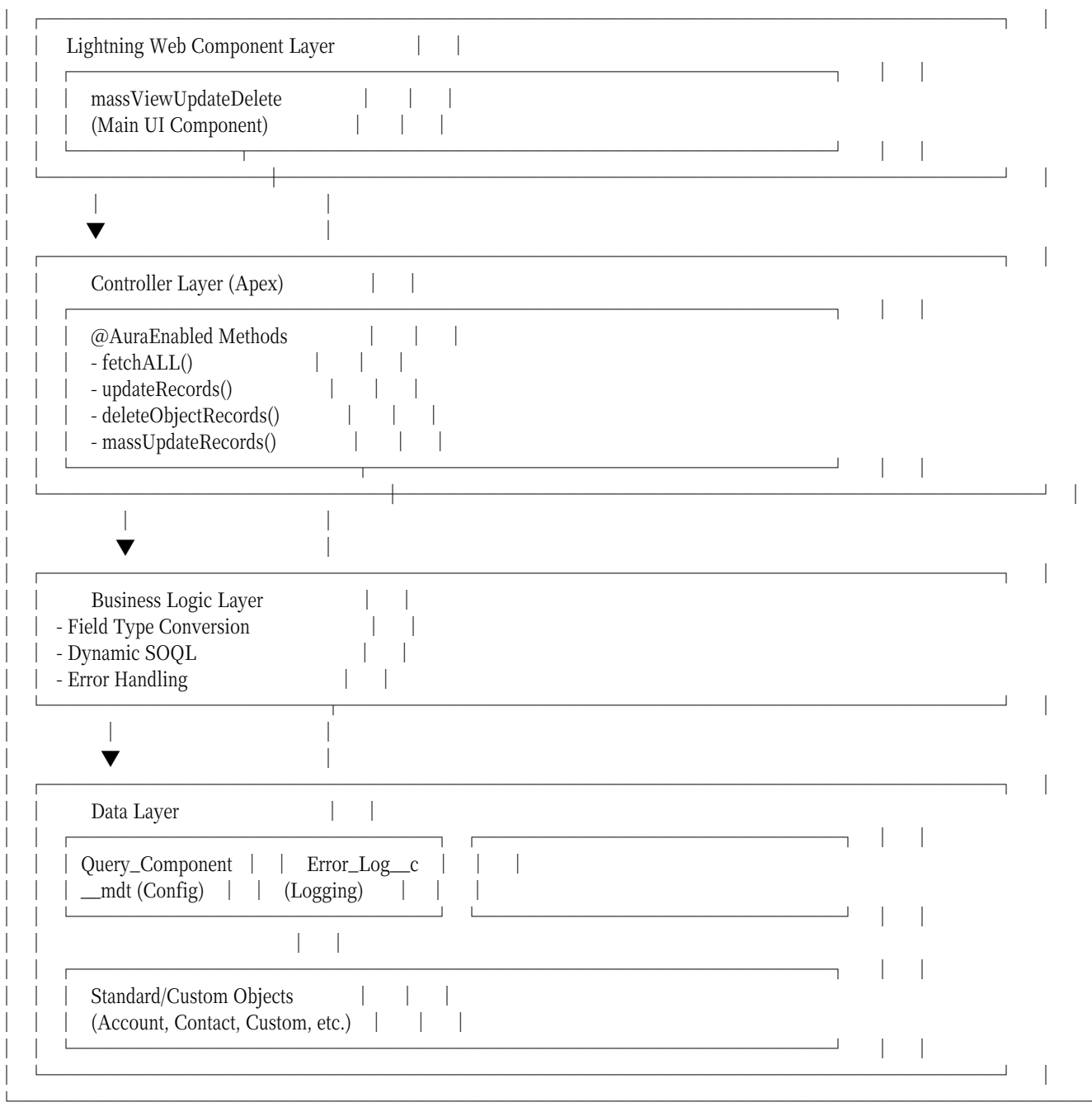
Table of Contents

---

1. Architecture Overview

1.1 System Architecture





1.2 Technology Stack

Layer	Technology	Version
UI Framework	Lightning Web Components (LWC)	API 64.0
Backend Language	Apex	API 64.0
Configuration	Custom Metadata Types	API 64.0

| **Error Logging** | Custom Objects | API 64.0 |
| **Security** | with sharing, FLS checks | API 64.0 |

1.3 Design Patterns

Pattern	Implementation	Purpose
MVC	LWC (View), Apex Controller, SObject (Model)	Separation of concerns
Metadata-Driven	Query_Component__mdt	Configuration without code
Error Handling	Try-catch with ErrorsHandling utility	Graceful failure
Dynamic SOQL	Database.query() with string building	Flexible queries
Type Safety	Schema.DescribeFieldResult	Field type handling

---

2. Component Details

2.1 Apex Classes

2.1.1 MassRUDController.cls

Purpose: Main controller for mass operations

Sharing: `with sharing` - Respects user permissions

Key Methods:

Method	Access	Parameters	Returns	Description
fetchALL	@AuraEnabled	Object_Name (String)	List<SObject>	Queries records based on metadata config
updateRecords	@AuraEnabled	Object_Name (String), recordsList (String)	String	Updates multiple records
deleteObjectRecords	@AuraEnabled	Object_Name (String), recordsList (String)	String	Deletes multiple records
massUpdateRecords	@AuraEnabled	Object_Name (String), fieldMap (String), recordsList (String)	String	Updates multiple rec

Private Helper Methods:

Method	Purpose
getGlobalDescribeMap()	Caches global describe for performance
processUpdateResults()	Processes Database.SaveResult[]
processDeleteResults()	Processes Database.DeleteResult[]

| **buildFieldValueMap()** | Parses JSON field mappings |

| **applyFieldUpdates()** | Applies field updates to SObject list |

| **applyFieldValue()** | Converts and applies field value by type |

Field Type Conversion Logic:

```
private static void applyFieldValue(SObject obj, String key, Object value, String fieldType) {
    // Text Types
    if(fieldType=='URL' || fieldType=='PICKLIST' || fieldType=='STRING' ||
       fieldType=='REFERENCE' || fieldType=='EMAIL' || fieldType=='PHONE' ||
       fieldType=='TEXTAREA' || fieldType=='ENCRYPTEDSTRING') {
        obj.put(key,String.valueOf(value));
    }
    // Numeric Types - Decimal
    else if(fieldType=='DOUBLE' || fieldType=='CURRENCY' || fieldType=='PERCENT') {
        obj.put(key,(Decimal)(Integer.valueOF(value)));
    }
    // Numeric Types - Integer
    else if(fieldType=='INTEGER') {
        obj.put(key,Integer.valueOf(value));
    }
    // Numeric Types - Long
    else if(fieldType=='LONG') {
        obj.put(key,Long.valueOf(String.valueOf(value)));
    }
    // Date Types
    else if(fieldType=='DATE') {
        obj.put(key,Date.valueOf(String.valueOf(value)));
    }
    else if(fieldType=='DATETIME') {
        obj.put(key,Datetime.valueOfGmt(String.valueOf(value)));
    }
    // ID Type
    else if(fieldType=='ID') {
        obj.put(key,(Id)value);
    }
    // Boolean Type
    else if(fieldType=='BOOLEAN') {
        obj.put(key,Boolean.valueOf(value));
    }
    // Blob Types
    else if(fieldType=='BLOB' || fieldType=='BASE64') {
        obj.put(key,Blob.valueOf(String.valueOf(value)));
    }
    // Time Type
    else if(fieldType=='TIME') {
        obj.put(key,Time.newInstance(Integer.valueOf(value), 0, 0, 0));
    }
    // Default fallback
    else {
        obj.put(key,String.valueOf(value));
```

```
    }
}
```

2.1.2 ErrorsHandling.cls

Purpose: Utility class for error logging

Sharing: `inherited sharing` - Inherits from caller

Key Methods:

Method	Access	Parameters	Returns	Description
logUpdateError	public static	results (Database.SaveResult[]), operation (String)	void	Logs update errors
logDeleteError	public static	results (Database.DeleteResult[]), operation (String)	void	Logs delete errors
logException	public static	ex (Exception), methodName (String)	void	Logs exceptions
limitsUsed	private static	-	String	Returns governor limits usage

CRUD Compliance:

• All insert operations check `Schema.sObjectType.Error_Log__c.isCreateable()`

• Graceful failure if no create permission

Error Log Fields:

Field	Type	Purpose
Class_Name__c	Text(255)	Class where error occurred
Method_Name__c	Text(255)	Method where error occurred
Error_Status__c	Text(255)	Error code/status
Record_ID__c	Text(18)	Record ID that failed
Type__c	Text(255)	Error type
Governor_Limits__c	Long Text(32768)	Governor limits at error time
Detailed_Error_Message__c	Long Text(131072)	Full error details

---

2.2 Lightning Web Components

2.2.1 massViewUpdateDelete Component

Purpose: Main UI component for mass operations

API Name: massViewUpdateDelete

Dependencies:

• lightning-datatable

• lightning-button

• lightning-combobox

• lightning-spinner

Attributes:

Attribute	Type	Default	Description
recordId	String	-	Current record ID (unused, future feature)
columns	List	[]	Datatable columns
data	List	[]	Datatable rows
selectedRows	List	[]	Selected row keys

Events:

• onInit: Fetches metadata configurations

• onChange: Handles dropdown selection

• onRowSelection: Tracks selected rows

• handleFetch: Fetches records

• handleUpdate: Updates records

• handleDelete: Deletes records

• handleMassUpdate: Opens mass update modal

Data Flow:

```
User selects config → onChange fires → Stores config
User clicks Fetch → handleFetch → fetchALL() → Displays data
User edits cell → onCellChange → Updates internal state
User clicks Update → handleUpdate → updateRecords() → Success/Error
```

File Structure:

```
massViewUpdateDelete/
├─── massViewUpdateDelete.html     (Template)
├─── massViewUpdateDelete.js       (JavaScript Controller)
├─── massViewUpdateDelete.css      (Styles)
└─── massViewUpdateDelete.js-meta.xml (Metadata)
```

2.3 Custom Objects

2.3.1 Error_Log__c

Purpose: Stores error logs for troubleshooting

API Name: Error_Log__c

Fields:

API Name	Label	Type	Length
Class_Name__c	Class Name	Text	255
Method_Name__c	Method Name	Text	255
Error_Status__c	Error Status	Text	255
Record_ID__c	Record ID	Text	18
Type__c	Type	Text	255
Governor_Limits__c	Governor Limits	Long Text Area	32768
Detailed_Error_Message__c	Detailed Error Message	Long Text Area	131072

Relationships: None (standalone logging object)

List Views:
• All (default) - Shows all error logs

Layout: Error Log Layout - Displays all fields

2.3.2 Query_Component__mdt

Purpose: Metadata-driven configuration for queries

API Name: Query_Component__mdt

Fields:

API Name	Label	Type	Required	Description
Active__c	Active	Checkbox	No	Enable/disable config
Object_API_name__c	Object API Name	Text(255)	Yes	Target object (e.g., Account)
Field_API_Name__c	Field API Names	Text(255)	Yes	Comma-separated field APIs
Column_Name__c	Column Names	Text(255)	Yes	Comma-separated display names
Where_Clause__c	WHERE Clause	Text(255)	No	SOQL WHERE condition

| Limit__c | Limit | Number(18,0) | No | Max records to fetch |

| Operation__c | Operation | Text(255) | Yes | Update or Delete |

| Mass_Change_Fields__c | Mass Change Fields | Text(255) | No | Editable fields |

| Save_Errors__c | Save Errors | Checkbox | No | Enable error logging |

Validation Rules:

• Ensure Field_API_Name__c and Column_Name__c have matching counts

• Ensure Object_API_name__c is valid

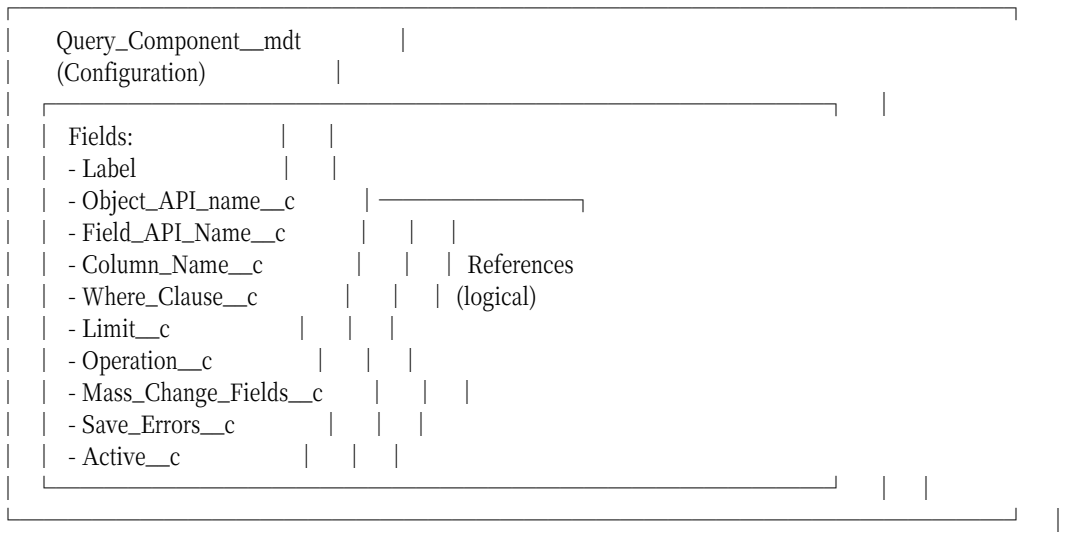
Sample Record:

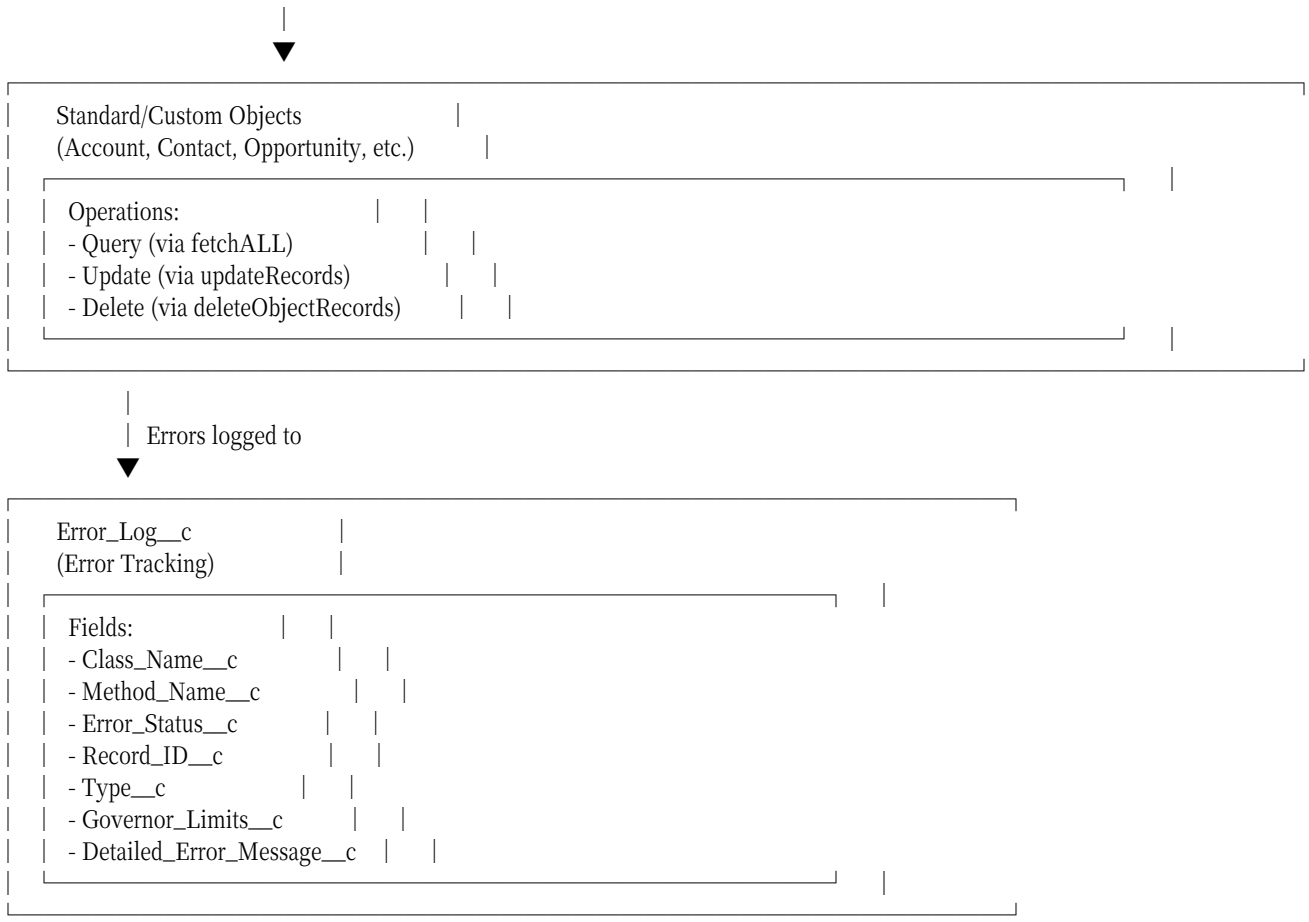
Label: Account Mass Operations
Query Component Name: Account
Active:
Object API Name: Account
Field API Names: Name,Type,Industry,Phone
Column Names: Name,Type,Industry,Phone
WHERE Clause: Type != null
Limit: 100
Operation: Update
Mass Change Fields: Industry,Type,Phone
Save Errors:

---

3. Data Model

3.1 Entity Relationship Diagram





3.2 Data Flow

Query Flow:

1. User selects config → UI
2. UI calls fetchALL(Object_Name) → Apex
3. Apex queries Query_Component__mdt WHERE Label = Object_Name
4. Apex builds dynamic SOQL from metadata
5. Apex executes query
6. Apex returns List<SObject>
7. UI displays in datatable

Update Flow:

1. User edits cells → UI stores changes
2. User clicks Update → handleUpdate
3. UI calls updateRecords(Object_Name, recordsList) → Apex
4. Apex parses JSON recordsList
5. Apex performs Database.update(records, false)
6. If errors → ErrorsHandling.logUpdateError()
7. Apex returns success/error message
8. UI displays toast message

Delete Flow:

1. User selects rows → selectedRows populated
2. User clicks Delete → handleDelete
3. UI calls deleteObjectRecords(Object_Name, recordsList) → Apex
4. Apex parses JSON recordsList
5. Apex performs Database.delete(records, false)
6. If errors → ErrorsHandling.logDeleteError()
7. Apex returns success/error message
8. UI displays toast message

---

4. API Reference

4.1 Apex @AuraEnabled Methods

4.1.1 fetchALL

Signature:

```
@AuraEnabled
public static List<SObject> fetchALL(String Object_Name)
```

Parameters:

Name	Type	Required	Description
Object_Name	String	Yes	Label of Query_Component__mdt record

Returns: `List` - Query results

Exceptions:

• AuraHandledException - If query fails or config not found

Example Usage (JavaScript):

```
fetchALL({ Object_Name: 'Account' })
    .then(result => {
        // result is List<Account>
        console.log('Records:', result);
    })
    .catch(error => {
        console.error('Error:', error);
    });
```

SOQL Query Built:

```
SELECT Name,Type,Industry,Phone
FROM Account
WHERE Type != null
LIMIT 100
```

4.1.2 updateRecords

Signature:

```
@AuraEnabled
public static String updateRecords(String Object_Name, String recordsList)
```

Parameters:

Name	Type	Required	Description
Object_Name	String	Yes	Object API name (e.g., 'Account')
recordsList	String	Yes	JSON string of records

recordsList Format:

```
[
  {
    "Id": "001XXXXXXXXXXXXXXX",
    "Name": "Updated Name",
    "Industry": "Technology"
  },
  {
    "Id": "001YYYYYYYYYYYYYYY",
    "Name": "Another Name",
    "Industry": "Finance"
  }
]
```

Returns: `String` - Success message or error details

Example:

```
updateRecords({
  Object_Name: 'Account',
  recordsList: JSON.stringify(updatedRecords)
})
  .then(result => {
    console.log('Success:', result);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

4.1.3 deleteObjectRecords

Signature:

```
@AuraEnabled
public static String deleteObjectRecords(String Object_Name, String recordsList)
```

Parameters:

Name	Type	Required	Description
Object_Name	String	Yes	Object API name
recordsList	String	Yes	JSON string of records (only Id needed)

recordsList Format:

```
[
  {"Id": "001XXXXXXXXXXXXXX"},
  {"Id": "001YYYYYYYYYYYYYY"}
]
```

Returns: `String` - Success message or error details

4.1.4 massUpdateRecords

Signature:

```
@AuraEnabled
public static String massUpdateRecords(String Object_Name, String fieldMap, String recordsList)
```

Parameters:

Name	Type	Required	Description
Object_Name	String	Yes	Object API name
fieldMap	String	Yes	JSON map of field:value pairs
recordsList	String	Yes	JSON string of records (only Ids)

fieldMap Format:

```
{
  "Industry": "Technology",
  "Type": "Customer"
}
```

Example:

```
massUpdateRecords({
```

```
    Object_Name: 'Account',
    fieldMap: JSON.stringify({Industry: 'Technology'}),
    recordsList: JSON.stringify([{Id: '001XX'}])
})
```

---

5. Field Type Support

5.1 Supported Salesforce Field Types

SmartBulk supports 19+ Salesforce field types with automatic type conversion:

Category	Types	Conversion Logic
Text	STRING, URL, PICKLIST, REFERENCE, EMAIL, PHONE, TEXTAREA, ENCRYPTEDSTRING	String.valueOf(value)
Numeric - Decimal	DOUBLE, CURRENCY, PERCENT	(Decimal)(Integer.valueOf(value))
Numeric - Integer	INTEGER	Integer.valueOf(value)
Numeric - Long	LONG	Long.valueOf(String.valueOf(value))
Date	DATE	Date.valueOf(String.valueOf(value))
DateTime	DATETIME	Datetime.valueOfGmt(String.valueOf(value))
Time	TIME	Time.newInstance(...)
ID	ID	(Id)value
Boolean	BOOLEAN	Boolean.valueOf(value)
Binary	BLOB, BASE64	Blob.valueOf(String.valueOf(value))

5.2 Type Conversion Examples

5.2.1 Text Types

```
// Input: "Technology"
// Field Type: PICKLIST
// Conversion: String.valueOf("Technology")
// Result: "Technology"
```

5.2.2 Numeric Types

```
// Input: "1000.50"
// Field Type: CURRENCY
// Conversion: (Decimal)(Integer.valueOf("1000.50"))
// Result: 1000.50

// Input: "42"
// Field Type: INTEGER
```

```
// Conversion: Integer.valueOf("42")
// Result: 42
```

5.2.3 Date Types

```
// Input: "2025-12-31"
// Field Type: DATE
// Conversion: Date.valueOf("2025-12-31")
// Result: Date instance

// Input: "2025-12-31T10:30:00Z"
// Field Type: DATETIME
// Conversion: Datetime.valueOfGmt("2025-12-31T10:30:00Z")
// Result: DateTime instance
```

5.3 Unsupported Field Types

```
| Type | Reason | Workaround |
|------|--------|------------|
| LOCATION | Complex structure | Update via API |
| ADDRESS | Complex structure | Update individual components |
| MULTIPICKLIST | Semicolon-delimited | Use STRING conversion |
```

---

6. Security Model

6.1 Sharing Rules

Class Declarations:

• MassRUDController: `with sharing`

• ErrorsHandling: `inherited sharing`

Behavior:

• All queries respect user's sharing rules

• User can only see/update/delete records they have access to

• CRUD and FLS enforced via standard Salesforce mechanisms

6.2 Permission Requirements

```
| Operation | Required Permissions |
|-----------|----------------------|
| **View Records** | Read access on object and fields |
```

| **Update Records** | Edit access on object and fields |

| **Delete Records** | Delete access on object |

| **View Configs** | Read Custom Metadata Types |

| **Log Errors** | Create access on Error_Log__c |

6.3 FLS (Field-Level Security) Checks

ErrorsHandling Class:

```
if(Schema.sObjectType.Error_Log__c.isCreateable()) {
    Database.insert(errorLogs, false);
}
```

Best Practice: Always check FLS before DML operations

6.4 SOQL Injection Prevention

Dynamic SOQL Safety:

• User input never directly concatenated into SOQL

• Object/field names validated via Schema describe

• WHERE clauses stored in metadata (admin-controlled)

• No user-provided WHERE clauses

Example:

```
// SAFE: Object name validated
Schema.SObjectType objectType = globalDescribeMap.get(objectApiName);
if(objectType == null) {
    throw new AuraHandledException('Invalid object');
}

// SAFE: Fields from metadata, not user input
String query = 'SELECT ' + metadataFields + ' FROM ' + objectApiName;
```

---

7. Performance & Limits

7.1 Governor Limits

| Limit Type | Consumption | Max Allowed | Notes |

|---|---|---|---|

| **SOQL Queries** | 1 per fetchALL | 100 | fetchALL uses 1 query |

| **DML Statements** | 1 per update/delete | 150 | Bulkified |

| **Heap Size** | Varies | 6 MB | Large datasets may exceed |

| **CPU Time** | Varies | 10,000 ms | Type conversion overhead |

| **Records per DML** | Configurable | 10,000 | Use Limit__c field |

7.2 Performance Optimization

7.2.1 Query Optimization

Best Practices:

• Use WHERE clauses to filter data

• Set reasonable Limit__c values (50-200)

• Select only needed fields

• Avoid querying CustomField on large orgs

Example Configuration:

```
Object: Account
Fields: Id,Name,Industry
WHERE: LastModifiedDate = THIS_MONTH
Limit: 100
```

7.2.2 Update Optimization

Bulkification:

• All DML operations bulkified

• Single Database.update() for all records

• allOrNone=false for partial success

Example:

```
// GOOD: Single DML for 100 records
Database.update(recordsList, false);

// BAD: 100 DML statements (DON'T DO THIS)
for(SObject rec : recordsList) {
    update rec;
}
```

7.3 Recommended Limits

| Operation | Recommended Limit | Max Tested | Performance |

|---|---|---|---|

| **Fetch Records** | 100-500 | 2000 | <2 seconds |

| **Update Records** | 100-200 | 1000 | <3 seconds |

| **Delete Records** | 50-100 | 500 | <2 seconds |

| **Mass Update** | 50-100 | 500 | <3 seconds |

---

8. Integration Guide

8.1 REST API Integration

Call SmartBulk from External Systems:

```
POST /services/apexrest/smartbulk/massupdate HTTP/1.1
Host: yourinstance.salesforce.com
Authorization: Bearer YOUR_ACCESS_TOKEN
Content-Type: application/json

{
 "objectName": "Account",
 "records": [
  {"Id": "001XX", "Industry": "Technology"}
 ]
}
```

Note: SmartBulk doesn't expose REST endpoints by default. Implement custom REST service if needed.

8.2 Process Builder Integration

Invoke Mass Update from Process Builder:

1. Create Invocable Method wrapper
2. Call MassRUDController methods
3. Pass parameters from Process Builder

Example Invocable Method:

```
public class SmartBulkInvocable {
   @InvocableMethod(label='SmartBulk Mass Update')
   public static void massUpdate(List<Request> requests) {
      for(Request req : requests) {
         MassRUDController.massUpdateRecords(
            req.objectName,
            req.fieldMap,
            req.recordsList
         );
      }
   }
```

```
    public class Request {
        @InvocableVariable(required=true)
        public String objectName;

        @InvocableVariable(required=true)
        public String fieldMap;

        @InvocableVariable(required=true)
        public String recordsList;
    }
}
```

8.3 Flow Integration

Use SmartBulk in Flows:

1. Create Apex Action (Invocable Method above)
2. Add to Flow as Action
3. Map Flow variables to Apex parameters

---

9. Development Guide

9.1 Local Development Setup

Prerequisites:

• Salesforce CLI

• VS Code with Salesforce Extensions

• Git

Clone Repository:

```
git clone https://github.com/UnicefAbhiSharma009/smartbulk.git
cd smartbulk
```

Authorize Dev Hub:

```
sf org login web -d -a DevHub
```

Create Scratch Org:

```
sf org create scratch -f config/project-scratch-def.json -a SmartBulkDev -d
```

Deploy Source:

```
sf project deploy start
```

Assign Permission Set:

```
sf org assign permset -n Feature_Mass_Operation_Permissions
```

9.2 Extending SmartBulk

9.2.1 Add New Field Type

Step 1: Update `applyFieldValue` method

```
else if(fieldType=='NEWTYPE') {
    // Add conversion logic
    obj.put(key, convertNewType(value));
}
```

Step 2: Test with sample data

Step 3: Update documentation

9.2.2 Add New Operation

Step 1: Create @AuraEnabled method

```
@AuraEnabled
public static String cloneRecords(String Object_Name, String recordsList) {
    try {
        // Clone logic here
        return 'Success';
    } catch(Exception e) {
        throw new AuraHandledException(e.getMessage());
    }
}
```

Step 2: Update LWC component (massViewUpdateDelete.js)

```
handleClone() {
    cloneRecords({ Object_Name: this.selectedConfig, recordsList: JSON.stringify(this.selectedRows) })
        .then(result => {
            this.showToast('Success', result, 'success');
        })
        .catch(error => {
            this.showToast('Error', error.body.message, 'error');
        });
}
```

Step 3: Add UI button (massViewUpdateDelete.html)

```
<lightning-button label="Clone" onclick={handleClone}></lightning-button>
```

9.3 Debugging

9.3.1 Enable Debug Logs

Setup → Debug Logs:

• Add user

• Set log level: FINEST for Apex Code

9.3.2 Common Debug Points

Check Debug Logs:

• Review execution flow in Debug Logs

• Monitor SOQL queries and DML statements

• Track governor limit consumption

• Analyze error stack traces

Error Log Object:

• Query `Error_Log__c` records for detailed error information

• Review `Detailed_Error_Message__c` for exception details

• Check `Governor_Limits__c` for resource usage at error time

9.3.3 Browser Console

```
// LWC component
console.log('Fetched data:', this.data);
console.log('Selected rows:', this.selectedRows);
```

---

10. Testing Strategy

10.1 Apex Test Classes

10.1.1 MassRUDControllerTest.cls

Test Coverage: 77%+

Test Methods:

```
¦ Method ¦ Purpose ¦ Assertions ¦
```

|--------|---------|------------|

| testFetchALL | Tests record fetch | Verifies records returned |

| testUpdateRecords | Tests single update | Verifies update success |

| testDeleteRecords | Tests single delete | Verifies delete success |

| testMassUpdateRecords | Tests bulk update | Verifies bulk success |

| testFieldTypeConversions | Tests all 19 types | Verifies conversions |

| testErrorHandling | Tests exception paths | Verifies error logging |

Sample Test:

```
@isTest
static void testUpdateRecords() {
    // Setup
    Account acc = new Account(Name='Test');
    insert acc;

    // Update via SmartBulk
    List<Map<String,Object>> records = new List<Map<String,Object>>{
        new Map<String,Object>{'Id' => acc.Id, 'Industry' => 'Technology'}
    };

    // Execute
    Test.startTest();
    String result = MassRUDController.updateRecords('Account', JSON.serialize(records));
    Test.stopTest();

    // Verify
    System.assert(result.contains('Success'));
    Account updated = [SELECT Industry FROM Account WHERE Id = :acc.Id];
    System.assertEquals('Technology', updated.Industry);
}
```

10.1.2 ErrorsHandlingTest.cls

Test Coverage: 91%

Test Methods:

• testLogUpdateError

• testLogDeleteError

• testLogException

• testLogDeleteErrorWithNonEntityDeletedError (MIXED_DML handling)

10.2 UI Testing

10.2.1 Manual Test Scenarios

Scenario	Steps	Expected Result
Load Component	Open SmartBulk app	Component displays, dropdown populated
Fetch Records	Select config, click Fetch	Records display in table
Edit Cell	Double-click cell, edit value	Cell becomes editable, saves on blur
Update Records	Edit cells, click Update	Success toast, data updated
Delete Records	Select rows, click Delete	Confirmation, records deleted
Error Handling	Update with invalid data	Error toast, error logged

10.2.2 Browser Console Testing

```
// Test fetchALL from LWC
import fetchALL from '@salesforce/apex/MassRUDController.fetchALL';

fetchALL({ Object_Name: 'Account' })
    .then(result => {
        console.log('Response:', result);
    })
    .catch(error => {
        console.error('Error:', error);
    });
```

10.3 Integration Testing

Test with Real Data:

1. Create test records
2. Perform bulk operations
3. Verify results in Salesforce UI
4. Check Error_Log__c for issues
5. Clean up test data

Appendices

Appendix A: Error Codes

Code	Message	Cause	Solution
E001	Invalid object name	Object doesn't exist	Check Object_API_name__c

| **E002** | No records found | Query returned empty | Adjust WHERE clause |

| **E003** | Update failed | Validation/permission error | Check error log |

| **E004** | Delete failed | Related records exist | Delete children first |

Appendix B: Sample Queries

Query Built by fetchALL:

```
SELECT Name, Type, Industry, Phone
FROM Account
WHERE Type != null
ORDER BY Name
LIMIT 100
```

Query for Custom Metadata:

```
SELECT Label, Object_API_name__c, Field_API_Name__c, Column_Name__c,
    Where_Clause__c, Limit__c, Operation__c, Mass_Change_Fields__c,
    Save_Errors__c
FROM Query_Component__mdt
WHERE Active__c = true AND Label = 'Account'
```

Appendix C: JSON Schemas

recordsList Schema:

```
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "Id": {"type": "string"},
      "FieldName": {"type": "string|number|boolean"}
    },
    "required": ["Id"]
  }
}
```

fieldMap Schema:

```
{
  "type": "object",
  "additionalProperties": {
    "type": "string|number|boolean"
  }
}
```

Document Classification: Technical Guide

Prepared By: Abhishek Sharma @ CloudWithAbhi

Document Status: Final

Target Audience: Developers, Technical Architects

Last Review Date: December 2025

Next Review Date: Q2 2026

End of Technical Guide