RetryTool - User Guide

Table of Contents

Overview

RetryTool is a Salesforce managed package that provides automatic retry functionality for handling transient errors like `UNABLE_TO_LOCK_ROW` across all Salesforce layers (LWC, Aura, Flow, Apex). This guide walks you through the complete setup, configuration, and maintenance procedures.

What RetryTool Solves

- **Row-locking conflicts** when multiple processes try to update the same record
- **Transient errors** in API callouts, timeouts, and system overload scenarios
- **Inconsistent error handling** across different Salesforce contexts
- **Manual retry attempts** that waste user time and reduce productivity

---

Prerequisites & Installation

System Requirements

1. **Lightning Web Security (LWS)** - **REQUIRED** for LWC integration
    - Navigate to Setup → Session Settings → Lightning Web Security
    - Ensure "Use Lightning Web Security for Lightning web components" is enabled
2. **Edition**: All editions supported (Developer, Professional, Enterprise, Unlimited)

Installation Steps

AppExchange Installation

1. **Access AppExchange**
    - Go to Salesforce AppExchange
    - Search for "RetryTool"

- Click "Get It Now"

2. **Choose Installation Link**
   - **Production**: Install Latest Version
   - **Sandbox**: Install Latest Version
3. **Installation Options**
   - **Recommended**: "Install for All Users"
   - **Alternative**: "Install for Admins Only" (requires manual permission assignment later)

Post-Installation Verification

After installation, verify these components were created automatically:

1. **Default Setting Record**: Navigate to RetryTool Settings tab
   - Record Name: "RetryTool_Default"
   - Developer Name: "RetryTool_Default"
   - Max Retry: 5
   - Waiting Time: 3000ms
   - Log Retention: 90 days
2. **Scheduled Job**: Navigate to Setup → Scheduled Jobs
   - Job Name: "RetryTool Log Cleanup Schedule"
   - Scheduled: Last day of each month at 11:00 PM GMT
3. **Permission Sets**: Setup → Permission Sets
   - `RetryTool_Admin`
   - `RetryTool_User`
   - `RetryTool_Debugger`

Initial Setup & Configuration

Step 1: Access RetryTool App

1. Click the **App Launcher** (9 dots icon)
2. Search for **"RetryTool"**
3. Click the RetryTool app to open it

You'll see three main tabs:

- **Retry Settings**: Configuration management
- **Retry Logs**: Operation monitoring
- **RetryTool Welcome**: Package overview

Step 2: Verify Default Configuration

Navigate to **Retry Settings** tab and open the **RetryTool_Default** record:

Basic Settings:

- **Developer Name**: RetryTool_Default (unique identifier)
- **Max Retry**: 5 (number of retry attempts)
- **Waiting Time**: 3000 (milliseconds between retries)
- **Log Retention In Days**: 90 (automatic cleanup period)

Advanced Settings:

- **Extra Retryable Errors**: Custom error patterns (pipe-separated)
- **Setting Link**: Formula field for easy navigation

Permission Management

Critical Requirement

⚠ RetryTool WILL NOT FUNCTION without proper permission assignment

⚠ **Production License Requirement**: In Production environments, RetryTool requires a **valid package license**. Without it, no retries will occur - all retryable errors like `UNABLE_TO_LOCK_ROW` will be treated as non-retryable.

Permission Set Overview

Permission Set	Use Case	Access Level
`RetryTool_Admin`	System administrators	Full CRUD on settings/logs, all Apex classes
`RetryTool_User`	End users needing retry functionality	Execute retry operations, read settings
`RetryTool_Debugger`	Troubleshooting support	Read-only access to logs and debug info

Assignment Methods

Method 1: Setup UI (Recommended)

1. **Navigate to Setup → Users**
2. **Click on user name**
3. **Click "Permission Set Assignments"**
4. **Click "Edit Assignments"**
5. **Add appropriate permission set** (`RetryTool_User` for most users)
6. **Save**

Method 2: Apex Code (Bulk Assignment)

```
// Bulk assign RetryTool_User to active users
// Note: For large orgs, consider adding LIMIT or WHERE conditions to
reduce query size
List<User> activeUsers = [SELECT Id FROM User WHERE IsActive = true];
PermissionSet ps = [SELECT Id FROM PermissionSet WHERE Name =
'RetryTool_User' LIMIT 1];

List<PermissionSetAssignment> assignments = new
List<PermissionSetAssignment>();
for (User u : activeUsers) {
    assignments.add(new PermissionSetAssignment(
        AssigneeId = u.Id,
        PermissionSetId = ps.Id
    ));
}
insert assignments;
```

Method 3: Data Loader (Large Organizations)

1. **Export user IDs** who need access
2. **Query PermissionSet ID**: `SELECT Id FROM PermissionSet WHERE Name = 'RetryTool_User' LIMIT 1`
3. **Create CSV** with columns: AssigneeId, PermissionSetId
4. **Use Data Loader** to insert PermissionSetAssignment records

---

Configuration Settings

Basic Settings Fields

- **Developer Name**: Unique identifier for this configuration
- **Max Retry**: Number of retry attempts (1-5)
- **Waiting Time**: Milliseconds between retries
- **Log Retention In Days**: How long to keep log records (1-365)

Default Settings Fallback

⚠ **Important**: If you use a setting name that doesn't exist, RetryTool automatically applies **default values**:

- **Max Retry**: 5
- **Waiting Time**: 3000ms
- **Log Retention**: 90 days

When this happens:

- After **sandbox refresh** if settings records are not restored
- When using **wrong setting names** in code
- During **testing** with non-existent settings

Extra Retryable Errors Field

Add custom error keywords (pipe-separated) to extend retry behavior beyond the built-in patterns.

Format: `KEYWORD1|KEYWORD2|KEYWORD3`

Example Keywords (for illustration only):

- `TIMEOUT`
- `OVERLOAD`
- `RATE_LIMIT`
- `NETWORK_ERROR`
- `CALLOUT_EXCEPTION`

Built-in Patterns (always active):

- `UNABLE_TO_LOCK_ROW`
- Multi-language record locking messages

Testing & Validation

Built-in Testing Tools

RetryTool includes comprehensive testing components accessible from any Retry Setting record:

1. LWC Testing Tab

Purpose: Test Lightning Web Component retry integration **Test Scenarios**:

- Persistent Retry Error (continuous failures)
- Non-Persistent Retry Error (intermittent failures)
- Non-Retry Error (validation of non-retryable errors)

Testing Steps:

1. Open any Retry Setting record
2. Click **LWC Tab**
3. Select error type from dropdown
4. Click **Test** button
5. Monitor retry behavior and logging

Custom Handlers (optional - you can do whatever you want in these functions):

```
retryTool.handleErrorRetry(this, error, callback, settingName, {
  onRetry: (currentRetry, maxRetry) => {
    console.log(`Retry ${currentRetry} of ${maxRetry}`);
    // You can do whatever you want here - update UI, show toasts, call
methods, etc.
  },
  onMaxRetriesExceeded: () => {
    console.error("Maximum attempts reached");
    // You can do whatever you want here - show error messages, enable
```

```
buttons, etc.
  },
  onRetryNotAllowed: () => {
    console.error("Cannot retry this error");
    // You can do whatever you want here - handle non-retryable errors,
navigate, etc.
  },
  processName: "YourComponent",
  processAction: "yourMethod"
});
```

⚠ **Important**: Replace `settingName` with the exact **Developer Name** from your RetryTool_Setting__c records. If the setting name doesn't exist, RetryTool will use default values (MaxRetry: 5, WaitingTime: 3000ms).

2. Aura Testing Tab

Purpose: Test Aura Component retry integration **Similar scenarios** as LWC testing **Legacy support** for existing Aura implementations

Custom Handlers (optional - you can do whatever you want in these functions):

```
helper.retryToolHandleErrorRetry(cmp, error, callback, settingName, {
  onRetry: function (currentRetry, maxRetry) {
    console.log("Retry " + currentRetry + " of " + maxRetry);
    // You can do whatever you want here - update UI, show toasts, call
methods, etc.
  },
  onMaxRetriesExceeded: function () {
    console.error("Maximum attempts reached");
    // You can do whatever you want here - show error messages, enable
buttons, etc.
  },
  onRetryNotAllowed: function () {
    console.error("Cannot retry this error");
    // You can do whatever you want here - handle non-retryable errors,
navigate, etc.
  },
  processName: "YourAuraComponent",
  processAction: "yourFunction"
});
```

⚠ **Important**: Replace `settingName` with the exact **Developer Name** from your RetryTool_Setting__c records. If the setting name doesn't exist, RetryTool will use default values (MaxRetry: 5, WaitingTime: 3000ms).

3. Screen Flow Testing Tab

Purpose: Test Flow DML fault path retry **Prerequisites**: Requires SObject Aggregator helper class - must be customized for your specific SObjects and include test class (examples provided below)

Custom Error Screens (optional - you can customize error handling as you wish):

- Use **OverrideErrorScreen** parameter to bypass default error messages
- Create your own error screen with custom messages, buttons, and navigation
- Add decision elements after RetryTool SubFlow to handle success/failure
- You can do whatever you want - show custom messages, redirect users, log errors, etc.

4. Auto-Launched Flow Testing Tab

Purpose: Test background Flow retry operations **Use Case**: Batch processing, scheduled flows, platform event handlers

Backend/Async Testing Examples:

Option 1: Flow DML Operation Action

- Drag "Retry Tool DML Operation" action into auto-launched flow
- Configure: DmlType (INSERT/UPDATE/DELETE), Records collection, RetryToolSetting name
- Test with bulk records for async retry behavior

⚠ **Important**: Use the exact **Developer Name** from your RetryTool_Setting__c records for the RetryToolSetting parameter.

Option 2: Direct Apex Integration

```
// Basic backend retry example
RetryToolDmlOperation.getInstance(records, 'BackendSetting').doUpdate();

// With process context for logging
RetryToolDmlOperation.getInstance(records, 'BatchProcessor')
    .setProcessName('DataMigrationBatch')
    .setProcessAction('updateRecords')
    .doInsert();
```

⚠ **Important**: Use the exact **Developer Name** from your RetryTool_Setting__c records. If the setting name doesn't exist, RetryTool will use default values (MaxRetry: 5, WaitingTime: 3000ms).

Async Test Scenarios:

- **Batch Jobs**: Test with 200+ records to trigger async processing
- **Platform Events**: Test retry behavior in event subscribers
- **Scheduled Operations**: Test retry in scheduled apex or flows

Creating Test Scenarios

Validation Rule Testing Method

Create temporary validation rule for testing:

1. **Navigate to Object Manager** → [Your Object] → Validation Rules
2. **Create New Rule**:
   - **Rule Name**: RetryTool_Test
   - **Error Condition Formula**: `!1=1 & ISCHANGED()` (Always run on update)
   - **Error Message**: `UNABLE_TO_LOCK_ROW` (or your custom error pattern)
3. **Test retry functionality**
4. **Deactivate rule** when testing complete

Custom Error Testing

Test custom error patterns:

1. **Add custom pattern** to Extra Retryable Errors: `CUSTOM_TEST_ERROR`
2. **Create validation rule** with error message: `CUSTOM_TEST_ERROR: Testing custom pattern`
3. **Test retry behavior**
4. **Verify logging** in Retry Logs tab

SObject Aggregator Helper Class

For Screen Flow integration, create this helper class customized for your SObjects:

Helper Class Example

```
public inherited sharing class RtSObjectAggregatorInvocable {
  @InvocableMethod(
    label='RetryTool - SObjectAggregatorInvocable'
    description='Invocable Method to aggregate sobjects for retry
operations'
    category='RetryTool'
  )
  public static List<OutputParam> doAggregation(List<InputParam> inputs) {
    List<OutputParam> results = new List<OutputParam>();

    for (InputParam input : inputs) {
      OutputParam output = new OutputParam();
      List<SObject> records = new List<SObject>();

      // Add your specific SObject types here
      if (input.account != null)
        records.add(input.account);
      if (input.accountList != null)
        records.addAll(input.accountList);
      if (input.contact != null)
        records.add(input.contact);
      if (input.contactList != null)
        records.addAll(input.contactList);
      // Add more SObject types as needed for your org

      output.sobjectAggregation = JSON.serialize(records);
```

```
      results.add(output);
    }

    return results;
  }

  public class InputParam {
    @InvocableVariable
    public Account account;
    @InvocableVariable
    public List<Account> accountList;
    @InvocableVariable
    public Contact contact;
    @InvocableVariable
    public List<Contact> contactList;
    // Add more SObject types as needed for your org
  }

  public class OutputParam {
    @InvocableVariable
    public String sobjectAggregation;
  }
}
```

Test Class Code Snippet

```
@IsTest
public class RtSObjectAggregatorInvocableTest {
  @IsTest
  static void testDoAggregation() {
    // Test data setup
    Account testAccount = new Account(Name = 'Test Account');
    insert testAccount;

    Contact testContact = new Contact(
      FirstName = 'Test',
      LastName = 'Contact',
      AccountId = testAccount.Id
    );
    insert testContact;

    // Prepare input parameters
    RtSObjectAggregatorInvocable.InputParam input = new
RtSObjectAggregatorInvocable.InputParam();
    input.account = testAccount;
    input.contact = testContact;
    input.accountList = new List<Account>{ testAccount };
    input.contactList = new List<Contact>{ testContact };

    // Execute the invocable method
    Test.startTest();
```

```
    List<RtSObjectAggregatorInvocable.OutputParam> results =
RtSObjectAggregatorInvocable.doAggregation(
      new List<RtSObjectAggregatorInvocable.InputParam>{ input }
    );
    Test.stopTest();

    // Verify results
    System.assertEquals(1, results.size(), 'Should return one result');
    System.assertNotEquals(
      null,
      results[0].sobjectAggregation,
      'Should have JSON output'
    );

    // Validate JSON serialization
    List<SObject> deserializedRecords = (List<SObject>) JSON.deserialize(
      results[0].sobjectAggregation,
      List<SObject>.class
    );
    System.assertEquals(
      4,
      deserializedRecords.size(),
      'Should contain all 4 records'
    );
  }

  @IsTest
  static void testEmptyInput() {
    // Test with empty input
    RtSObjectAggregatorInvocable.InputParam input = new
RtSObjectAggregatorInvocable.InputParam();

    Test.startTest();
    List<RtSObjectAggregatorInvocable.OutputParam> results =
RtSObjectAggregatorInvocable.doAggregation(
      new List<RtSObjectAggregatorInvocable.InputParam>{ input }
    );
    Test.stopTest();

    // Verify empty result handling
    System.assertEquals(1, results.size(), 'Should return one result');
    System.assertNotEquals(
      null,
      results[0].sobjectAggregation,
      'Should have JSON output'
    );
  }
}
```

Important Notes:

- Modify the `InputParam` class to include your organization's specific SObject types

- Update the aggregation logic to handle your custom objects
- Ensure test class covers all your SObject types for proper deployment coverage

---

Sandbox Refresh Procedures

Problem Statement

When refreshing a sandbox from production:

- **RetryTool package** remains installed
- **Configuration data** (`RetryTool_Setting__c` records) is **NOT automatically refreshed**
- **Log data** (`RetryTool_Log__c` records) is **NOT carried over**
- **Permission assignments** may need to be recreated

⚠ **Critical**: If settings records are not restored, RetryTool will use **default values** (MaxRetry: 5, WaitingTime: 3000ms, LogRetention: 90 days) for all operations until settings are recreated.

Post-Refresh Setup Checklist

Step 1: Verify Package Installation

1. **Navigate to Setup → Installed Packages**
2. **Verify RetryTool package** is listed as installed
3. **Check package version** matches expected version

Step 2: Recreate Configuration Data

Option A: Manual Recreation

1. **Document settings from Production**:
    - Navigate to RetryTool Settings tab in Production
    - Open each setting record and note down configuration values:
        - Developer Name
        - Max Retry
        - Waiting Time
        - Log Retention In Days
        - Extra Retryable Errors
2. **Create settings in Sandbox**:
    - Navigate to RetryTool Settings tab in Sandbox
    - Create new records matching production configuration
    - Verify Developer Names match exactly

Option B: Data Loader Method

1. **Export from Production**:
    - Use Data Loader to export `RetryTool_Setting__c` records

- Include fields: `DeveloperName__c`, `MaxRetry__c`, `WaitingTime__c`, `LogRetentionInDays__c`, `ExtraRetryableErrors__c`

2. **Import to Sandbox**:
   - Use Data Loader to import the exported records to Sandbox
   - Verify all records imported successfully

Step 3: Verify Scheduled Jobs

Check if log cleanup job is scheduled:

1. **Navigate to Setup** → Scheduled Jobs
2. **Look for**: "RetryTool Log Cleanup Schedule"
3. **If missing**, run post-install script manually:

```
// Execute in Developer Console
RetryToolLogCleanupSchedule.scheduleBatch(
    'RetryTool Log Cleanup Schedule',
    '0 0 23 L * ?'  // Default cron: 11 PM on last day of month
);
```

Step 4: Reassign Permissions

Sandbox user permissions may not include RetryTool access:

1. **Identify users** who need RetryTool access
2. **Assign permission sets** using Setup UI or bulk methods (see Permission Management section)
3. **Test functionality** with assigned users

Step 5: Validation Testing

Run comprehensive testing to ensure functionality:

1. **Use built-in testers** (LWC, Aura, Flow tabs)
2. **Verify error detection** and retry behavior
3. **Check logging functionality**
4. **Confirm settings are working** correctly

Package Uninstallation

Pre-Uninstallation Requirements

⚠ **CRITICAL**: Remove all RetryTool references before uninstalling the package to avoid metadata dependency errors.

Step 1: Identify Dependencies

Search for RetryTool references in your org using Setup UI:

1. **Apex Classes**:
    - Navigate to Setup → Apex Classes
    - Use Global Search to find "RetryTool" references
    - Review each class that contains RetryTool usage
2. **Lightning Components**:
    - Navigate to Setup → Lightning Components
    - Search for components containing "czywks_rt" namespace references
    - Check both LWC and Aura components
3. **Flows**:
    - Navigate to Setup → Flows
    - Search for flows containing "RetryTool" actions or elements
    - Review Flow definitions for RetryTool invocable actions

Step 2: Remove Code Dependencies

Lightning Web Components

Remove RetryTool imports and usage:

```
// REMOVE these lines:
import { retryTool } from "czywks_rt/retryTool";

// REMOVE retry handling code:
retryTool.handleErrorRetry(
    this, error, callback, settingName, processContext
);

// REPLACE with standard error handling:
.catch(error => {
    console.error('Error:', error);
    // Your standard error handling
});
```

Aura Components

Remove RetryTool interface and component:

```
<!-- REMOVE these lines: -->
implements="czywks_rt:RetryToolAura"
<czywks_rt:retryTool aura:id="retryToolCmpId" />
```

```
// REMOVE helper method calls:
helper.retryToolHandleErrorRetry(cmp, error, callback, settingName,
context);

// REPLACE with standard error handling
```

Flows

Remove RetryTool Invocable Actions:

1. **Open each Flow** that uses RetryTool
2. **Remove these elements**:
    - "RetryTool DML Operation" actions
    - "RetryTool SObject Aggregator" actions
    - Fault path connections to RetryTool subflows
3. **Replace with standard DML operations**
4. **Save and activate** updated flows

Apex Code

Remove RetryTool class usage:

```
// REMOVE these lines:
RetryToolDmlOperation.getInstance(records, 'SettingName').doInsert();

// REPLACE with standard DML:
try {
    insert records;
} catch (DmlException e) {
    // Your standard error handling
    throw e;
}
```

Step 3: Remove Custom Helper Classes

If you created custom helper classes (like SObject Aggregator):

1. **Identify helper classes** created for RetryTool integration (including test classes)
2. **Remove or comment out** RetryTool-specific functionality
3. **Update for your specific SObject DML operations** or remove entirely
4. **Deploy changes** to remove dependencies

Step 4: Validate Removal

Test your org to ensure all functionality works without RetryTool:

1. **Run existing unit tests**

2. **Test critical business processes**
3. **Verify no compilation errors**
4. **Check Flow execution**

Step 5: Uninstall Package

Only after removing all dependencies:

1. **Navigate to Setup** → Installed Packages
2. **Find RetryTool** package
3. **Click "Uninstall"**
4. **Follow uninstallation wizard**
5. **Confirm removal**

Step 6: Post-Uninstallation Cleanup

After successful uninstallation:

1. **Remove permission set assignments** (they'll be invalid)
2. **Clean up any remaining references** in documentation
3. **Update deployment scripts** if they referenced RetryTool
4. **Notify team members** of the removal

Troubleshooting

Common Issues

Issue: "RetryTool functionality not working"

Cause: License expiration in Production environment **Solution**:

- **Production Only**: Check package license expiration - without valid license, all errors are treated as non-retryable
- Verify license is current and not expired

Issue: "Permission denied" or "Access not allowed"

Cause: Permission sets not assigned **Solution**: Verify user has `RetryTool_User` or `RetryTool_Admin` permission set assigned

Issue: "Lightning Web Security error"

Cause: LWS not enabled for LWC integration **Solution**: Enable LWS in Setup → Session Settings → Lightning Web Security

Issue: "Settings not found" or "Using default values"

Cause: Developer Name mismatch, missing settings, or wrong setting name in code **Solution**:

- Verify setting exists with exact Developer Name used in code

- Check if settings were restored after sandbox refresh
- Remember: RetryTool uses default values (MaxRetry: 5, WaitingTime: 3000ms) when settings don't exist

Issue: "Logs not being created"

Cause: Platform Event trigger not working or permission issues **Solution**: Check trigger is active and user has permission to publish platform events

Issue: "Scheduled job not running"

Cause: Log cleanup job not scheduled or failed **Solution**: Check Scheduled Jobs and reschedule if necessary

Issue: "Retries not working in Production but work in Sandbox"

Cause: Invalid package license in Production environment
Solution: Ensure valid RetryTool license in Production - **without valid license, RetryTool will NOT retry** and will treat all `UNABLE_TO_LOCK_ROW` and other retryable errors as non-retryable

Debug Mode

Enable debug mode for troubleshooting:

1. **Assign `RetryTool_Debugger` permission set**
2. **Check Debug Logs** for RetryTool namespace activity
3. **Monitor Platform Events** for logging issues
4. **Review Scheduled Job History** for cleanup issues

Support Channels

- **Internal Documentation**: RetryTool App → Welcome tab
- **Built-in Testing**: Use testing tabs for validation
- **Salesforce Logs**: Monitor debug logs for detailed error messages
- **Package Documentation**: Review technical documentation in docs/ folder

---

Appendix

Default Settings Reference

```
{
  "DeveloperName__c": "RetryTool_Default",
  "MaxRetry__c": 5,
  "WaitingTime__c": 3000,
  "LogRetentionInDays__c": 90,
  "ExtraRetryableErrors__c": null
}
```

Permission Sets Summary

Object	RetryTool_Admin	RetryTool_User	RetryTool_Debugger
RetryTool_Setting__c	Full CRUD	Read	Read
RetryTool_Log__c	Full CRUD	Create/Read	Read
RetryTool_Log_PE__e	Create/Read	Create/Read	Read
Apex Classes	All classes	Essential classes	Debug classes

Cron Expression Reference

Default log cleanup schedule: `0 0 23 L * ?`

- **0 0 23**: 11:00 PM
- **L**: Last day of month
- *****: Every month
- **?**: Any day of week

Custom schedule examples:

- Daily at 2 AM: `0 0 2 * * ?`
- Weekly Sunday 3 AM: `0 0 3 ? * SUN`
- First day of month 1 AM: `0 0 1 1 * ?`