Bank Integration - Complete Documentation

Airwallex to ERPNext Integration

Akhilam Inc

November 06, 2025

1 Bank Integration - Complete Documentation

Airwallex to ERPNext Integration

2 Introduction & Quick Start

Seamless integration between Airwallex and ERPNext for automatic financial transaction synchronization.

2.1 Overview

The Bank Integration App provides comprehensive integration between Airwallex and ERPNext, enabling automatic synchronization of financial transactions. This app supports multiple Airwallex clients, scheduled syncing, intelligent currency matching, robust error handling, and advanced transaction filtering.

2.2 Features

- **Multi-Client Support**: Manage multiple Airwallex clients from a single setting
- **Scheduled Syncing**: Automatic hourly, daily, weekly, or monthly transaction sync
- **Manual Sync**: Sync historical transactions for specific date ranges
- **Smart Token Caching**: Database-backed token storage with automatic refresh
- **Currency Matching**: Intelligent bank account assignment based on currency
- **Transaction Type Filtering**: Include or exclude specific transaction types
- **Duplicate Prevention**: Automatic detection and skipping of existing transactions
- **Real-time Progress**: Live updates during sync operations
- **Comprehensive Logging**: Detailed error logs with privacy protection
- **Background Processing**: Long-running syncs handled by background jobs
- **Paginated API Calls**: Efficient handling of large transaction volumes

2.3 Installation

You can install this app using the bench CLI:

```
cd $PATH_TO_YOUR_BENCH
bench get-app https://github.com/Akhilam-Inc/erpnext_airwallex_integra
bench install-app bank_integration
```

2.3.1 Post-Installation Setup

1. Navigate to **Bank Integration Setting** in your ERPNext instance
2. Configure your Airwallex API URL (e.g., `https://api.airwallex.com/api/v1`)
3. Add your Airwallex clients with their credentials and bank accounts
4. Configure transaction type filters (optional)
5. Enable the integration and set up your sync schedule
6. Test authentication to verify connectivity

For detailed setup instructions, see the Configuration Guide.

2.4 Quick Start

2.4.1 Basic Configuration

1. Go to **Bank Integration Setting**
2. Enable Airwallex Integration
3. Set API URL: `https://api.airwallex.com/api/v1`
4. Add Airwallex Client:
   - Client ID
   - API Key
   - Linked Bank Account (must match currency)
5. Configure Transaction Filters (optional):
   - Add filters to include/exclude specific transaction types
   - Examples: Include only PAYMENT and REFUND, or Exclude FEE and ADJUSTMENT
6. Click **Test Authentication**
7. Set Sync Schedule (Hourly/Daily/Weekly/Monthly)
8. Save

2.4.2 Transaction Type Filtering

Control which transaction types are imported:

Include Only Payments and Refunds:

```
Transaction Type: PAYMENT     → Action: Include
Transaction Type: REFUND      → Action: Include
```

Exclude Fees and Adjustments:

```
Transaction Type: FEE         → Action: Exclude
Transaction Type: ADJUSTMENT  → Action: Exclude
```

Available Transaction Types: - PAYMENT, REFUND, DEPOSIT, WITHDRAWAL - FEE, ADJUSTMENT, TRANSFER - CONVERSION_SELL, CONVERSION_BUY - PAYOUT, DISPUTE_REVERSAL - And 20+ additional types

2.4.3 Sync Transactions

Scheduled Sync (Automatic): - Runs based on your configured schedule - Syncs from last sync date to current time - Applies configured transaction type filters - No user intervention required

Manual Sync (Historical): 1. Enable “Sync Old Transactions” 2. Set From Date and To Date 3. Click **Start Sync** 4. Monitor progress in real-time

2.5 Documentation

Complete documentation is available in the doc/ directory:

2.5.1 Getting Started

- **Overview** - Introduction and key features
- **Configuration Guide** - Detailed setup instructions

2.5.2 Workflow Documentation

- **Scheduled Sync Workflow** - Automatic periodic syncing
- **Manual Sync Workflow** - Historical data sync
- **Common Sync Process** - Core sync logic

2.5.3 Technical Details

- **Authentication & Token Management** - Token caching and security
- **Data Mapping** - Field transformations and filtering
- **Error Handling & Recovery** - Troubleshooting guide

Start here: doc/README.md

2.6 Architecture

2.6.1 Core Components

1. **Bank Integration Setting** - Global configuration and multi-client management
2. **Transaction Type Filter** - Configurable filtering for specific transaction types
3. **Scheduler** - Cron-triggered sync functions for scheduled operations
4. **Transaction Sync** - Main orchestration and client-specific processing
5. **API Layer** - Base API client with authentication and token management
6. **Data Mapping** - Airwallex to ERPNext field transformation

2.6.2 Code Structure

```
bank_integration/
├── airwallex/                          # Airwallex integration logic
│   ├── api/                           # API client layer
│   │   ├── base_api.py               # Base API client
│   │   ├── airwallex_authenticator.py # Authentication
│   │   └── financial_transactions.py  # Transactions API
│   ├── scheduler.py                   # Cron job functions
│   ├── transaction.py                 # Sync orchestration
│   └── utils.py                       # Data mapping utilities
├── bank_integration/                   # Main module
│   └── doctype/                       # DocTypes
│       ├── bank_integration_setting/  # Settings DocType
│       ├── bank_integration_log/      # Log DocType
│       ├── airwallex_client/          # Client child table
│       └── transaction_type_filter/   # Transaction filtering
└── fixtures/                          # Custom field definitions
```

2.7 Key Concepts

2.7.1 Sync Types

Scheduled Sync: - Automatic, incremental, triggered by cron - Date range calculated from last sync date - Runs in background without user intervention - Applies configured transaction type filters

Manual Sync: - User-initiated, specific date range - Background job with real-time progress - Useful for historical data import - Applies configured transaction type filters

2.7.2 Transaction Filtering

- **Whitelist Approach**: Use "Include" filters to sync only specific transaction types
- **Blacklist Approach**: Use "Exclude" filters to skip specific transaction types
- **No Filters**: All transaction types are synced (default behavior)
- **Filter Precedence**: Include filters take precedence over Exclude filters

2.7.3 Authentication

- Database-cached tokens with automatic refresh
- Multi-client support with individual credentials
- Automatic retry on authentication failures (401)
- 5-minute buffer before token expiry

2.7.4 Data Mapping

- **Currency Matching**: Bank account only assigned if transaction currency matches account currency
- **Auto-Classification**: Deposit vs withdrawal based on amount sign
- **Duplicate Prevention**: Checks `transaction_id` field before creating
- **Transaction Filtering**: Applied before mapping to reduce processing overhead
- **Comprehensive Mapping**: All relevant Airwallex fields mapped to ERPNext

2.7.5 Sync Status States

Status	Description
Not Started	Initial state, no sync has been initiated
In Progress	Sync is currently running
Completed	Sync finished successfully with no errors
Completed with Errors	Sync finished but some transactions had errors
Failed	Sync failed completely

2.8 API Reference

2.8.1 Main Functions

2.8.1.1 `sync_transactions(from_date, to_date, setting_name)`

Main sync orchestrator that processes all configured clients.

Parameters: - `from_date` (str/datetime): Start date for transaction sync - `to_date` (str/datetime): End date for transaction sync - `setting_name` (str): Name of Bank Integration Setting document

2.8.1.2 `sync_client_transactions(client, from_date_iso, to_date_iso, settings)`

Syncs transactions for a specific Airwallex client.

Parameters: - `client` (AirwallexClient): Client configuration object - `from_date_iso` (str): ISO8601 formatted start date - `to_date_iso` (str): ISO8601 formatted end date - `settings` (BankIntegrationSetting): Settings document

2.8.1.3 `should_sync_transaction(transaction_type)`

Determines if a transaction type should be synced based on configured filters.

Parameters: - `transaction_type` (str): Airwallex transaction type

Returns: - `bool`: True if transaction should be synced, False if filtered out

2.8.1.4 `map_airwallex_to_erpnext(txn, bank_account)`

Maps Airwallex transaction to ERPNext Bank Transaction format.

Parameters: - `txn` (dict): Airwallex transaction payload - `bank_account` (str): ERPNext Bank Account name

Returns: - `dict`: ERPNext Bank Transaction document dictionary

For detailed API documentation, see Data Mapping.

2.9 Troubleshooting

2.9.1 Common Issues

Authentication Failed - Verify Client ID and API Key are correct - Check API URL is set to `https://api.airwallex.com/api/v1` - Ensure credentials have proper permissions in Airwallex

No Transactions Syncing - Check if transaction type filters are too restrictive - Review Bank Integration Log for "filtered out" messages - Verify transaction types exist in your Airwallex data - Ensure sync status is not stuck in "In Progress"

Wrong Transactions Syncing - Review your transaction type filter strategy (Include vs Exclude) - Check for mixed Include/Exclude filters (Include takes precedence) - Verify transaction type names match Airwallex values exactly

Bank Account Not Set - Check transaction currency matches bank account currency - Verify bank account is properly configured in ERPNext - Review Data Mapping documentation

Duplicate Transactions - App automatically prevents duplicates based on `transaction_id` - Check Bank Transaction list for existing transactions - Review error logs for duplicate entry errors

For comprehensive troubleshooting, see Error Handling & Recovery.

2.10 Development

2.10.1 Contributing

This app uses `pre-commit` for code formatting and linting. Please install pre-commit and enable it for this repository:

```
cd apps/bank_integration
pre-commit install
```

Pre-commit is configured to use the following tools: - **ruff** - Python linting and formatting - **eslint** - JavaScript linting - **prettier** - Code formatting - **pyupgrade** - Python syntax modernization

2.10.2 Running Tests

```
# Run all tests
bench --site your-site run-tests --app bank_integration

# Run specific test file
bench --site your-site run-tests --app bank_integration --module bank_
```

2.10.3 CI/CD

This app uses GitHub Actions for continuous integration:

- **CI**: Installs this app and runs unit tests on every push to `develop` branch
- **Linters**: Runs Frappe Semgrep Rules and pip-audit on every pull request

2.11 Support

For issues, questions, or contributions: - **GitHub Issues**: Report an issue - **Documentation**: Review the guides in the `doc/` directory - **Troubleshooting**: See Error Handling & Recovery

2.12 Roadmap

- [x] **Transaction Type Filtering** - Configurable include/exclude filters
- [ ] Support for additional payment gateways
- [ ] Enhanced transaction filtering (amount, date, currency ranges)
- [ ] Automatic bank reconciliation
- [ ] Multi-currency conversion support

- [ ] Advanced reporting and analytics
- [ ] Webhook support for real-time updates

2.13 License

Made with by Akhilam Inc

3 Overview

3.1 Introduction

The Bank Integration App is a Frappe/ERPNext application that integrates with Airwallex to synchronize bank transactions automatically. This app enables seamless financial data flow from Airwallex into ERPNext’s Bank Transaction doctype.

3.2 Key Features

- **Multi-Client Support**: Configure multiple Airwallex clients with different credentials and bank accounts
- **Automatic Scheduled Syncing**: Set up periodic syncs (Hourly, Daily, Weekly, Monthly)
- **Manual Historical Sync**: Sync old transactions within a specified date range
- **Smart Token Management**: Database-cached tokens with automatic refresh
- **Duplicate Prevention**: Intelligent duplicate detection to avoid redundant entries
- **Currency Matching**: Automatic bank account assignment based on currency match
- **Real-time Progress Tracking**: Live updates during sync operations
- **Comprehensive Error Handling**: Graceful error handling with detailed logging

3.3 Sync Types

3.3.1 1. Scheduled Sync

Automatic periodic synchronization triggered by Frappe’s scheduler based on configured schedule: - **Hourly**: Syncs last 2 hours (or from last sync date) -**Daily**: Syncs yesterday (or from last sync date) - **Weekly**: Syncs last 7 days (or from last sync date) -**Monthly**: Syncs last 30 days (or from last sync date)

3.3.2 2. Old Transactions Sync

Manual synchronization for historical transactions: - User-defined date range (From/To dates) - Runs in background job queue - One-hour timeout for long-running operations - Progress tracking via UI

3.4 Architecture

```
bank_integration/
├── airwallex/
│   ├── api/
│   │   ├── base_api.py              # Base API class with auth & toke
management
│   │   ├── airwallex_authenticator.py  # Authentication handler
│   │   └── financial_transactions.py   # Financial transactions API
│   ├── scheduler.py                 # Cron job handlers
│   ├── transaction.py               # Sync logic
│   └── utils.py                     # Data mapping utilities
└── bank_integration/
    └── doctype/
        ├── bank_integration_setting/    # Main configuration doctype
        ├── airwallex_client/            # Client configuration (child
table)
        └── bank_integration_log/        # Sync logs
```

3.5 Data Flow

1. **Scheduler/User** triggers sync operation
2. **Sync Manager** orchestrates the sync process
3. **API Client** fetches transactions from Airwallex
4. **Token Manager** handles authentication automatically
5. **Data Mapper** transforms Airwallex data to ERPNext format
6. **Transaction Creator** inserts Bank Transaction records
7. **Progress Tracker** updates status and notifies user

3.6 Next Steps

- Configuration Guide - How to set up the integration
- Scheduled Sync Workflow - Detailed scheduled sync process
- Manual Sync Workflow - Detailed manual sync process
- Authentication & Token Management - Token caching strategy
- Data Mapping - How Airwallex data is mapped to ERPNext
- Error Handling - Error handling and recovery

4 Configuration Guide

4.1 Bank Integration Setting

The Bank Integration Setting is a Single DocType that holds all configuration for the Airwallex integration.

4.1.1 Access

Navigate to: **Bank Integration** workspace → **Bank Integration Setting**

4.1.2 Configuration Fields

4.1.2.1 Integration Settings

Field	Type	Description
`enable_airwallex`	Checkbox	Enable/disable the Airwallex integration
`api_url`	Data	Airwallex API base URL
`enable_log`	Checkbox	Enable detailed API logging

4.1.2.2 Token Management (Auto-managed)

Field	Type	Description
`token`	Small Text	Cached authentication token (auto-populated)
`token_expiry`	Datetime	Token expiration time (auto-populated)

4.1.2.3 Scheduled Sync Settings

Field	Type	Description
sync_schedule	Select	Schedule frequency: Hourly, Daily, Weekly, Monthly
last_sync_date	Datetime	Last successful sync timestamp (auto-updated)

4.1.2.4 Manual Sync Settings

Field	Type	Description
sync_old_transactions	Checkbox	Enable manual sync for historical transactions
from_date	Datetime	Start date for manual sync
to_date	Datetime	End date for manual sync

4.1.2.5 Sync Status (Auto-managed)

Field	Type	Description
sync_status	Select	Current sync status: Not Started, In Progress, Completed, Completed with Errors, Failed
processed_records	Int	Number of transactions processed (auto-updated)
total_records	Int	Total transactions to process (auto-updated)
sync_progress	Percent	Sync progress percentage (auto-updated)

4.1.2.6 Airwallex Clients (Child Table)

Field	Type	Description
airwallex_client_id	Data	Client ID from Airwallex (required)
airwallex_api_key	Password	API Key from Airwallex (required)
bank_account	Link	ERPNext Bank Account to map transactions to (required)
token	Small Text	Client-specific cached token (auto-managed)
token_expiry	Datetime	Client-specific token expiry (auto-managed)

4.1.2.7 Transaction Type Filters (Child Table)

Field	Type	Description
transaction_type	Select	Airwallex transaction type to filter
filter_action	Select	Include or Exclude this transaction type

Available Transaction Types: - `PAYMENT` - Payment transactions - `REFUND` - Refund transactions - `DEPOSIT` - Deposit transactions - `WITHDRAWAL` - Withdrawal transactions - `FEE` - Fee transactions - `ADJUSTMENT` - Account adjustment transactions - `TRANSFER` - Transfer transactions - `CONVERSION_SELL` - Currency conversion sell transactions - `CONVERSION_BUY` - Currency conversion buy transactions - `PAYOUT` - Payout transactions - `DISPUTE_REVERSAL` - Dispute reversal transactions - And 18 additional types (see Transaction Mapping Reference for complete list)

4.2 Transaction Type Filtering

4.2.1 How It Works

The transaction type filtering system allows you to control which Airwallex transaction types are imported into ERPNext.

4.2.1.1 Filtering Strategies

1. Whitelist Approach (Include Filters) - Add "Include" filters for transaction types you want to sync - Only the specified types will be imported - All other types will be skipped

2. Blacklist Approach (Exclude Filters) - Add "Exclude" filters for transaction types you don't want to sync - All types except the excluded ones will be imported

3. No Filters - If no filters are configured, all transaction types are imported - This is the default behavior for backward compatibility

4.2.1.2 Filter Examples

Example 1: Only sync payments and refunds

```
Transaction Type: PAYMENT,     Action: Include
Transaction Type: REFUND,      Action: Include
```

Result: Only PAYMENT and REFUND transactions are imported.

Example 2: Exclude fees and adjustments

```
Transaction Type: FEE,         Action: Exclude
Transaction Type: ADJUSTMENT,  Action: Exclude
```

Result: All transaction types except FEE and ADJUSTMENT are imported.

Example 3: Mixed filters (Not recommended)

```
Transaction Type: PAYMENT,     Action: Include
Transaction Type: FEE,         Action: Exclude
```

Result: Since there's an Include filter, only PAYMENT transactions are imported (whitelist takes precedence).

4.2.1.3 Best Practices

1. **Use one strategy**: Either use all Include filters or all Exclude filters, not both
2. **Test thoroughly**: After setting up filters, run a small manual sync to verify behavior
3. **Monitor logs**: Check Bank Integration Logs to see which transactions are filtered out
4. **Start permissive**: Begin with no filters, then add exclusions as needed

4.3 Setup Steps

4.3.1 1. Initial Configuration

1. Navigate to **Bank Integration Setting**
2. Set the **API URL** (e.g., `https://api.airwallex.com/api/v1`)
3. Optionally enable **Enable Log** for detailed logging

4.3.2 2. Add Airwallex Clients

For each Airwallex account you want to sync:

1. Click **Add Row** in the Airwallex Clients table
2. Enter **Airwallex Client ID**
3. Enter **Airwallex API Key**
4. Select the corresponding **Bank Account** in ERPNext
5. Click **Save**

Important: The bank account currency should match the currencies of transactions from that Airwallex client.

4.3.3 3. Configure Transaction Type Filters (Optional)

To control which transaction types are imported:

1. In the **Transaction Type Filters** section, click **Add Row**
2. Select a **Transaction Type** from the dropdown
3. Choose **Action**: Include or Exclude
4. Repeat for additional transaction types
5. Click **Save**

Note: If no filters are configured, all transaction types will be imported.

4.3.4 4. Test Authentication

1. Click the **Test Authentication** button
2. The system will verify credentials for all configured clients
3. Green checkmarks indicate successful authentication
4. Red errors indicate failed authentication (fix credentials and try again)

4.3.5 5. Enable Integration

1. Check **Enable Airwallex**
2. Click **Save**

Note: The system automatically tests authentication when you enable the integration. If authentication fails, the integration will be automatically disabled.

4.3.6 6. Configure Scheduled Sync

1. Select **Sync Schedule** (Hourly/Daily/Weekly/Monthly)
2. Click **Save**

The scheduler will automatically start syncing based on your schedule.

4.3.7 7. Configure Manual Sync (Optional)

For syncing historical transactions:

1. Check **Sync Old Transactions**
2. Set **From Date** and **To Date**
3. Click **Save**
4. Click **Start Sync** button that appears

4.4 Validation Rules

- At least one Airwallex client must be configured
- Client ID, API Key, and Bank Account are required for each client
- Authentication must succeed before enabling the integration
- From Date must be before To Date for manual sync
- Sync cannot start if another sync is in progress
- Transaction type filters must have both transaction type and action specified

4.5 Best Practices

1. **Test Authentication First**: Always test authentication before enabling
2. **Match Currencies**: Ensure bank account currency matches transaction currencies
3. **Configure Filters Early**: Set up transaction type filters before your first sync
4. **Start Small**: Begin with a short date range for manual sync to test
5. **Monitor Progress**: Watch the sync progress percentage during operation
6. **Check Logs**: Review Bank Integration Logs for any errors
7. **Review Filtered Transactions**: Check logs to see which transactions are being filtered out
8. **Credential Security**: API keys are stored encrypted as password fields

4.6 Troubleshooting Transaction Filters

4.6.1 Common Issues

No transactions syncing after adding filters - Check if you have Include filters that are too restrictive - Verify transaction types exist in your Airwallex data - Review sync logs for “filtered out” messages

Wrong transactions syncing - Review your filter strategy (Include vs Exclude) - Check for typos in transaction type names - Ensure you’re using the correct Airwallex transaction type values

Mixed Include/Exclude behavior - When Include filters exist, Exclude filters are ignored - Use consistent filtering strategy (all Include or all Exclude)

4.6.2 Debugging Steps

1. **Check logs**: Look for “filtered out” messages in ERPNext logs
2. **Test without filters**: Temporarily remove all filters to see all available transaction types
3. **Use small date range**: Test with 1-2 days of data to see filtering behavior
4. **Review Airwallex data**: Check what transaction types your Airwallex account actually generates

5 Scheduled Sync Workflow

5.1 Overview

Scheduled sync automatically syncs transactions at regular intervals using Frappe’s scheduler. This is the recommended approach for keeping transactions up-to-date.

5.2 Workflow Diagram

```
graph TB
    Start([Scheduler Trigger]) --> CheckEnabled{Is
Airwallex<br/>Enabled?}
    CheckEnabled -->|No| EndDisabled([End: Disabled])
    CheckEnabled -->|Yes| CheckSchedule{Matches<br/>Schedule Setting?}
    CheckSchedule -->|No| EndNoMatch([End: Wrong Schedule])
    CheckSchedule -->|Yes| CheckInProgress{Sync Already<br/>In
Progress?}
    CheckInProgress -->|Yes| EndInProgress([End: Already Running])
    CheckInProgress -->|No| SetStatus[Set Status: In Progress]

    SetStatus --> CheckLastSync{Last Sync<br/>Date Exists?}
    CheckLastSync -->|Yes| UseLastSync[Start Date = Last Sync
Date<br/>End Date = Now]
    CheckLastSync -->|No| CalcSchedule[Calculate Date Range:
<br/>Hourly: -2 hours<br/>Daily: -1 day<br/>Weekly: -7
days<br/>Monthly: -30 days]

    UseLastSync --> StartSync[Start Sync Process]
    CalcSchedule --> StartSync
    StartSync --> ProcessClients[Process All Clients]
    ProcessClients --> UpdateStatus{Any<br/>Errors?}

    UpdateStatus -->|No| StatusComplete[Status: Completed]
    UpdateStatus -->|Yes| StatusWithErrors[Status: Completed with
Errors]

    StatusComplete --> UpdateLastSync[Update Last Sync Date to Now]
    StatusWithErrors --> UpdateLastSync
    UpdateLastSync --> End([End: Success])

    style Start fill:#90EE90
    style End fill:#90EE90
    style EndDisabled fill:#FFD700
    style EndNoMatch fill:#FFD700
    style EndInProgress fill:#FFD700
```

5.3 Scheduler Functions

Located in `bank_integration/airwallex/scheduler.py`:

5.3.1 `run_hourly_sync()`

- Triggered every hour by Frappe scheduler
- Syncs last 2 hours of transactions (first run) or from last sync date
- Best for high-transaction-volume accounts

5.3.2 `run_daily_sync()`

- Triggered once per day (typically at midnight)
- Syncs yesterday's transactions (first run) or from last sync date
- Best for moderate transaction volumes

5.3.3 `run_weekly_sync()`

- Triggered once per week
- Syncs last 7 days (first run) or from last sync date
- Best for low-transaction-volume accounts

5.3.4 `run_monthly_sync()`

- Triggered once per month
- Syncs last 30 days (first run) or from last sync date
- Best for very low-transaction-volume accounts

5.4 Execution Flow

5.4.1 1. Trigger Phase

```
def run_hourly_sync():
    setting = frappe.get_single("Bank Integration Setting")

    # Check conditions
    if setting.enable_airwallex and \
       setting.sync_schedule == "Hourly" and \
       setting.sync_status != "In Progress":
        sync_scheduled_transactions("Bank Integration Setting", "Hourl
```

5.4.2 2. Date Calculation Phase

```
def sync_scheduled_transactions(setting_name, schedule_type):
    setting = frappe.get_single("Bank Integration Setting")

    # Prevent concurrent runs
    if setting.sync_status == "In Progress":
        return

    setting.db_set('sync_status', 'In Progress')

    end_date = frappe.utils.now_datetime()

    # Use incremental sync if possible
    if setting.last_sync_date:
        start_date = frappe.utils.get_datetime(setting.last_sync_date)
    else:
        # First run - calculate based on schedule
        if schedule_type == "Hourly":
            start_date = end_date - timedelta(hours=2)
        elif schedule_type == "Daily":
            start_date = end_date - timedelta(days=1)
        # ... etc
```

5.4.3 3. Sync Execution Phase

- Calls `sync_transactions(start_date, end_date, setting_name)`
- Processes all configured clients
- See Common Sync Process for details

5.4.4 4. Completion Phase

```
# On success
setting.db_set('last_sync_date', frappe.utils.now())
setting.db_set('sync_status', 'Completed')
```

5.5 Advantages of Scheduled Sync

1. **Automatic**: No manual intervention required
2. **Incremental**: Only syncs new transactions since last run
3. **Efficient**: Smaller date ranges mean faster processing
4. **Reliable**: Runs even if no one is logged in
5. **Consistent**: Predictable execution times

5.6 Configuration via hooks.py

Schedules are configured in `bank_integration/hooks.py`:

```
scheduler_events = {
    "hourly": [
        "bank_integration.airwallex.scheduler.run_hourly_sync"
    ],
    "daily": [
        "bank_integration.airwallex.scheduler.run_daily_sync"
    ],
    "weekly": [
        "bank_integration.airwallex.scheduler.run_weekly_sync"
    ],
    "monthly": [
        "bank_integration.airwallex.scheduler.run_monthly_sync"
    ]
}
```

5.7 Monitoring Scheduled Syncs

5.7.1 Via UI

- Check **Sync Status** in Bank Integration Setting
- View **Last Sync Date** to confirm recent execution
- Monitor **Processed Records** counter

5.7.2 Via Logs

- Check **Bank Integration Log** for sync history
- Review **Error Log** for failed syncs
- Check Frappe scheduler logs: `bench --site [site-name] doctor`

5.8 Troubleshooting

5.8.1 Scheduler Not Running

```
# Check if scheduler is enabled
bench --site [site-name] doctor

# Enable scheduler
bench --site [site-name] enable-scheduler

# Check scheduler status
bench --site [site-name] show-scheduler-status
```

5.8.2 Sync Stuck "In Progress"

- Manually reset status in Bank Integration Setting
- Or click **Restart Sync** button

5.8.3 Missed Syncs

- Check if Airwallex integration is enabled
- Verify sync schedule matches setting
- Check error logs for authentication failures

5.9 Best Practices

1. **Choose Appropriate Schedule**: Match to your transaction volume
2. **Monitor Initial Runs**: Watch first few syncs to ensure proper operation
3. **Review Logs Regularly**: Check for errors or warnings
4. **Keep Credentials Updated**: Renew API keys before expiry
5. **Test After Changes**: Use manual sync to test after configuration changes

6 Manual Sync Workflow

6.1 Overview

Manual sync (Old Transactions Sync) allows you to sync historical transactions for a specific date range. This is useful for: - Initial data migration - Backfilling missed transactions - Re-syncing specific periods - Testing the integration

6.2 Workflow Diagram

```
graph TB
    Start([User Clicks Start Sync]) --> ValidateDates{From/To
Dates<br/>Valid?}
    ValidateDates -->|No| EndInvalid([End: Show Error])
    ValidateDates -->|Yes| CheckInProgress{Sync Already<br/>In
Progress?}
    CheckInProgress -->|Yes| EndInProgress([End: Already Running])
    CheckInProgress -->|No| ResetCounters[Reset Status & Counters:
<br/>Status = In Progress<br/>Processed = 0<br/>Total = 0<br/>Progress
= 0%]

    ResetCounters --> EnqueueJob[Enqueue Background Job:<br/>Queue:
long<br/>Timeout: 1 hour<br/>Function: sync_transactions]
    EnqueueJob --> ShowMessage[Show User Message:<br/>'Sync job
started']
    ShowMessage --> BackgroundStart[Background Job Starts]

    BackgroundStart --> ProcessSync[Execute Sync Process]
    ProcessSync --> UpdateProgress[Real-time Progress Updates]
    UpdateProgress --> CheckComplete{Sync<br/>Complete?}

    CheckComplete -->|No| UpdateProgress
    CheckComplete -->|Yes| CheckErrors{Any<br/>Errors?}

    CheckErrors -->|No| StatusComplete[Status: Completed]
    CheckErrors -->|Yes| StatusWithErrors[Status: Completed with
Errors]

    StatusComplete --> UpdateLastSync[Update Last Sync Date]
    StatusWithErrors --> UpdateLastSync
    UpdateLastSync --> Notify[Publish Realtime Notification]
    Notify --> End([End: Show Summary])

    style Start fill:#90EE90
    style End fill:#90EE90
    style EndInvalid fill:#FFB6C6
    style EndInProgress fill:#FFD700
```

6.3 Step-by-Step Process

6.3.1 1. User Configuration

1. Navigate to **Bank Integration Setting**
2. Enable **Sync Old Transactions** checkbox
3. Set **From Date** (start of date range)
4. Set **To Date** (end of date range)
5. Click **Save**
6. Click **Start Sync** button (appears after saving)

6.3.2 2. Validation

The system validates: - Both From Date and To Date are provided - From Date is before or equal to To Date - No other sync is currently in progress - At least one Airwallex client is configured

6.3.3 3. Job Enqueueing

```python
@frappe.whitelist()
def start_transaction_sync(self):
    # Update status
    self.db_set('sync_status', 'In Progress')
    self.db_set('processed_records', 0)
    self.db_set('total_records', 0)
    self.db_set('sync_progress', 0)
    self.db_set('last_sync_date', frappe.utils.now())

    # Enqueue background job
    enqueue(
        'bank_integration.airwallex.transaction.sync_transactions',
        queue='long',
        timeout=3600,  # 1 hour
        from_date=str(self.from_date),
        to_date=str(self.to_date),
        setting_name=self.name
    )
```

6.3.4 4. Background Execution

The job runs in the background, allowing the user to: - Continue working in ERPNext - Close the browser - Monitor progress from Bank Integration Setting page

6.3.5 5. Progress Tracking

Real-time updates are published via `frappe.publish_realtime()`:

```
// UI receives updates
frappe.realtime.on('transaction_sync_progress', (data) => {
    // Update progress bar
    // Show processed/total counts
    // Display current status
});
```

6.3.6 6. Completion

Upon completion, the user receives a notification with: - Total transactions processed - Number of transactions created - Number of duplicates skipped - Number of errors encountered

6.4 UI Actions

6.4.1 Start Sync Button

- Appears when `sync_old_transactions` is checked
- Initiates the background sync job
- Disabled while sync is in progress

6.4.2 Restart Sync Button

- Appears when sync is complete or failed
- Resets status and starts a new sync
- Useful for retrying failed syncs

6.4.3 Stop Sync Button

- Appears while sync is in progress
- Marks sync as stopped
- Background job may take time to actually stop

6.5 Date Range Considerations

6.5.1 Large Date Ranges

- May take significant time to process
- Risk of timeout (1-hour limit)
- Consider breaking into smaller ranges

6.5.2 Recommended Ranges

- **First-time sync**: 1-3 months at a time
- **Regular backfill**: 1-2 weeks at a time
- **Testing**: 1-7 days

6.5.3 Overlapping Dates

- Safe to sync overlapping date ranges
- Duplicate detection prevents redundant entries
- Useful for ensuring no gaps

6.6 Monitoring Progress

6.6.1 Via UI

Watch real-time updates in Bank Integration Setting: -**Sync Status**: Current status - **Processed Records**: Count of processed transactions - **Total Records**: Estimated total (may update during sync) - **Sync Progress**: Percentage complete

6.6.2 Via Logs

- **Bank Integration Log**: Detailed sync events
- **Error Log**: Any errors encountered

- **Bank Transaction**: Created transaction records

6.7 Error Recovery

6.7.1 Common Errors

Error	Cause	Solution
“From and To dates are required”	Missing dates	Set both dates before starting
“From date cannot be greater than To date”	Invalid date range	Correct the date order
“No Airwallex clients configured”	No clients set up	Add at least one client
“Sync already in progress”	Concurrent sync attempt	Wait for current sync to complete
“Authentication failed”	Invalid credentials	Check and update API credentials
“Timeout”	Date range too large	Use smaller date range

6.7.2 Recovery Steps

1. **If Sync Fails**:
   - Check Error Log for details
   - Fix the underlying issue
   - Click **Restart Sync**
2. **If Sync Hangs**:
   - Wait for timeout (1 hour)
   - Manually reset status to “Not Started”
   - Click **Restart Sync**
3. **If Partial Sync**:
   - Note last successful transaction date
   - Adjust From Date to continue from there
   - Start new sync

6.8 Best Practices

1. **Test First**: Start with a small date range (1-7 days) to test
2. **Break Up Large Ranges**: Sync 1-3 months at a time for initial migration
3. **Monitor Actively**: Watch first few syncs to catch issues early
4. **Check Duplicates**: Review that duplicate detection is working
5. **Verify Currency Matching**: Ensure transactions map to correct bank accounts
6. **Document Your Process**: Note date ranges synced for audit trail
7. **Schedule Off-Hours**: Run large syncs during low-usage periods

6.9 Comparison with Scheduled Sync

Aspect	Manual Sync	Scheduled Sync
Trigger	User action	Automatic (cron)
Date Range	User-specified	Auto-calculated
Execution	Background job	Direct execution
Use Case	Historical data	Ongoing sync
Timeout	1 hour	No timeout
Frequency	On-demand	Regular intervals

7 Common Sync Process

7.1 Overview

Both scheduled and manual syncs converge on a common sync process that handles the actual data synchronization. This document details that shared process.

7.2 Process Flow Diagram

```
graph TB
    Start([sync_transactions called]) --> ConvertDates[Convert Dates
to ISO8601 Format]
    ConvertDates --> CheckClients{Airwallex Clients<br/>Configured?}
    CheckClients -->|No| ThrowError[Throw Error:<br/>No Clients
Configured]
```

```
configured]
    CheckClients -->|Yes| LoopClients[Loop Through Each Client]

    LoopClients --> SyncClient[sync_client_transactions]
    SyncClient --> InitAPI[Initialize FinancialTransactions
API<br/>with Client Credentials]
    InitAPI --> EnsureAuth[Ensure Authentication Headers]

    EnsureAuth --> CheckAuthHeader{Authorization<br/>Header Exists?}
    CheckAuthHeader -->|Yes| MakeAPICall[Make API Call:<br/>GET
/financial_transactions]
    CheckAuthHeader -->|No| GetToken[Get Valid Token from DB/API]
    GetToken --> MakeAPICall

    MakeAPICall --> CheckResponse{API Response<br/>Status?}
    CheckResponse -->|401 Unauthorized| RefreshToken[Auto-refresh
Token & Retry]
    RefreshToken --> CheckRetry{Retry<br/>Success?}
    CheckRetry -->|No| LogAPIError[Log API Error]
    CheckRetry -->|Yes| ProcessTxns[Process Transactions]
    CheckResponse -->|Success 200| ProcessTxns
    CheckResponse -->|Error 4xx/5xx| LogAPIError

    ProcessTxns --> LoopTxns[Loop Through Transactions]
    LoopTxns --> CheckDuplicate{Transaction ID<br/>Exists in DB?}
    CheckDuplicate -->|Yes| SkipTxn[Skip Transaction<br/>Increment
Skipped Counter]
    CheckDuplicate -->|No| MapData[Map Airwallex → ERPNext]

    MapData --> CheckCurrency{Transaction Currency<br/>== Bank Account
Currency?}
    CheckCurrency -->|Yes| SetBankAccount[Set Bank Account]
    CheckCurrency -->|No| BlankBankAccount[Leave Bank Account Blank]

    SetBankAccount --> CreateDoc[Create Bank Transaction Doc]
    BlankBankAccount --> CreateDoc
    CreateDoc --> InsertDoc[doc.insert]

    InsertDoc --> CheckInsert{Insert<br/>Success?}
    CheckInsert -->|Success| IncrementCreated[Increment Created
Counter]
    CheckInsert -->|Duplicate Error| SkipTxn
    CheckInsert -->|Other Error| LogTxnError[Log Error<br/>Increment
Error Counter]

    SkipTxn --> UpdateProgress[Update Progress<br/>Publish Realtime
Event]
    IncrementCreated --> UpdateProgress
    LogTxnError --> UpdateProgress

    UpdateProgress --> CheckMoreTxns{More<br/>Transactions<br/>in
Page?}
    CheckMoreTxns -->|Yes| LoopTxns
    CheckMoreTxns -->|No| CheckMorePages{Has<br/>More Pages?}

    CheckMorePages -->|Yes| NextPage[Increment page_num]
    NextPage --> MakeAPICall
    CheckMorePages -->|No| ClientComplete[Client Sync Complete]

    ClientComplete --> CheckMoreClients{More<br/>Clients?}
    CheckMoreClients -->|Yes| LoopClients
    CheckMoreClients -->|No| FinalUpdate[Calculate Final Status]

    FinalUpdate --> CheckErrors{Any<br/>Errors?}
    CheckErrors -->|Yes| StatusWithErrors[Status: Completed with
Errors]
    CheckErrors -->|No| StatusCompleted[Status: Completed]

    StatusWithErrors --> UpdateLastSync[Update Last Sync Date]
    StatusCompleted --> UpdateLastSync
    UpdateLastSync --> PublishComplete[Publish Completion
Notification]
    PublishComplete --> End([End: Sync Complete])

    LogAPIError --> FailedStatus[Set Status: Failed]
    ThrowError --> FailedStatus
    FailedStatus --> EndFailed([End: Sync Failed])

    style Start fill:#90EE90
    style End fill:#90EE90
    style EndFailed fill:#FFB6C6
    style CheckDuplicate fill:#FFA500
    style CheckCurrency fill:#FFA500
```

7.3 Function Breakdown

7.3.1 `sync_transactions(from_date, to_date, setting_name)`

Purpose: Main orchestrator for syncing transactions

Parameters: - `from_date`: Start date for sync - `to_date`: End date for sync - `setting_name`: Name of Bank Integration Setting doc (typically "Bank Integration Setting")

Process: 1. Load Bank Integration Setting 2. Validate clients are configured 3. Convert dates to ISO8601 format 4. Loop through each configured client 5. Call `sync_client_transactions()` for each 6. Aggregate results (processed, created, errors) 7. Update final status 8. Update last sync date

Returns: Implicitly via status updates

7.3.2 `sync_client_transactions(client, from_date_iso, to_date_iso, settings)`

Purpose: Sync transactions for a specific Airwallex client

Parameters: - `client`: Airwallex Client child doc - `from_date_iso`: ISO8601 formatted start date - `to_date_iso`: ISO8601 formatted end date - `settings`: Bank Integration Setting doc

Process: 1. Initialize `FinancialTransactions` API client 2. Authenticate (get/verify token) 3. Paginate through API results 4. Process each transaction: - Check for duplicates - Map to ERPNext format - Validate currency match - Create Bank Transaction doc 5. Handle errors gracefully 6. Update progress periodically

Returns: `(processed, created)` tuple

7.3.3 `transaction_exists(transaction_id)`

Purpose: Check if transaction already exists

Parameters: - `transaction_id`: Airwallex transaction ID

Process:

```
return frappe.db.exists("Bank Transaction", {"transaction_id": transac
```

Returns: Boolean

7.4 Pagination Handling

```
page_num = 0
page_size = 100

while True:
    response = api.get_list(
        from_created_at=from_date_iso,
        to_created_at=to_date_iso,
        page_num=page_num,
        page_size=page_size
    )

    if not response or not response.get('items'):
        break

    transactions = response.get('items', [])
    has_more = response.get('has_more', False)

    # Process transactions...

    if not has_more:
        break

    page_num += 1
```

Key Points: - Fetches 100 transactions per page - Continues until `has_more` is False - Prevents memory issues with large datasets

7.5 Progress Tracking

```
setting.update_sync_progress(total_processed, total_processed, "In Pro

# Inside update_sync_progress():
progress = (processed / total * 100) if total > 0 else 0
self.db_set('processed_records', processed)
self.db_set('total_records', total)
self.db_set('sync_progress', progress)
self.db_set('sync_status', status)

frappe.publish_realtime(
    'transaction_sync_progress',
    {'processed': processed, 'total': total, 'progress': progress},
    user=frappe.session.user
)
```

Purpose: Keep user informed of progress in real-time

7.6 Counters

The sync process tracks multiple counters:

Counter	Description
`total_processed`	Total transactions examined
`total_created`	Transactions successfully created
`total_skipped`	Duplicates skipped
`total_errors`	Transactions that failed

Calculation:

```
total_processed = total_created + total_skipped + total_errors
```

7.7 Transaction Processing Loop

```
for txn in transactions:
    try:
        transaction_id = txn.get('id')

        # Duplicate check
        if transaction_exists(transaction_id):
            total_skipped += 1
            total_processed += 1
            continue

        # Map data
        erpnext_txn = map_airwallex_to_erpnext(txn, client.bank_accour

        # Validate
        if not erpnext_txn.get('transaction_id'):
            total_errors += 1
            total_processed += 1
            continue

        # Create doc
        bank_txn_doc = frappe.get_doc(erpnext_txn)
        bank_txn_doc.insert()

        total_created += 1
        total_processed += 1

    except frappe.DuplicateEntryError:
        total_skipped += 1
        total_processed += 1

    except Exception as e:
        total_errors += 1
        total_processed += 1
        frappe.log_error(traceback.format_exc(), f"Transaction Error -
```

Key Features: - Errors don't stop the sync - Each transaction counted once - Detailed error logging - Graceful duplicate handling

7.8 Status Determination

```
final_status = "Completed" if total_errors == 0 else "Completed with
setting.update_sync_progress(total_processed, total_processed, final_s
```

Logic: - **Completed**: No errors occurred - **Completed with Errors**: Some transactions failed but sync finished - **Failed**: Sync couldn't complete (set elsewhere)

7.9 Final Summary

```
summary_msg = (
    f"Transaction sync completed. "
    f"Processed: {total_processed}, Created: {total_created}, "
    f"Skipped: {total_skipped}, Errors: {total_errors}"
)

frappe.logger().info(summary_msg)
bi_log.create_log(summary_msg)

frappe.publish_realtime(
    'transaction_sync_complete',
    {
        'processed': total_processed,
        'created': total_created,
        'skipped': total_skipped,
        'errors': total_errors,
        'status': final_status,
        'message': f"Sync completed. Created {total_created}, skipped
    },
    user=frappe.session.user
)
```

Delivery: - Console log - Bank Integration Log - Real-time notification to user - Status fields in Bank Integration Setting

7.10 Performance Considerations

7.10.1 Batch Processing

- API pagination reduces memory usage
- Processes 100 transactions at a time
- Updates progress every 10 transactions

7.10.2 Database Optimization

- Single duplicate check per transaction
- Bulk status updates (not per-transaction)
- Commit frequency balanced for performance

7.10.3 Error Isolation

- Single transaction error doesn't stop sync
- Client error doesn't stop other clients
- Comprehensive try-catch blocks

7.11 Best Practices

1. **Monitor Progress**: Watch real-time updates during first few syncs
2. **Review Logs**: Check Bank Integration Log after completion
3. **Validate Results**: Spot-check created Bank Transactions
4. **Check Counters**: Ensure processed = created + skipped + errors
5. **Investigate Errors**: Review Error Log for failed transactions
6. **Test Scenarios**: Test with various date ranges and data volumes

8 Authentication & Token Management

8.1 Overview

The Bank Integration App uses a sophisticated token management system that: - Caches tokens in the database for persistence - Automatically refreshes expired tokens - Handles 401 Unauthorized errors gracefully - Minimizes authentication requests

8.2 Token Caching Strategy

8.2.1 Database Storage

Tokens are stored in the database rather than in-memory cache: -**Advantage**: Persist across server restarts - **Advantage**: Shared across multiple workers/processes - **Advantage**: Better auditability - **Location**: `Bank Integration Setting` doctype

8.2.2 Token Fields

```
token = DF.Small Text  # The bearer token
token_expiry = DF.Datetime  # When the token expires
```

8.3 Authentication Flow

```
sequenceDiagram
    participant API as API Client
    participant Base as Base API
    participant Auth as Authenticator
    participant DB as Database
    participant AW as Airwallex API

    API->>Base: Make API Request
    Base->>Base: ensure_authenticated_headers()

    alt Authorization header exists
        Base->>API: Use existing token
    else No authorization header
        Base->>DB: Check cached token & expiry

        alt Token valid (not expired)
            DB-->>Base: Return cached token
            Base->>Base: Set Authorization header
            Base->>API: Proceed with request
        else Token expired or missing
            Base->>Auth: authenticate()
            Auth->>AW: POST /api/v1/authentication/login
            AW-->>Auth: Return token & expires_at
            Auth->>DB: Save token & expiry
            DB-->>Auth: Confirmed
            Auth-->>Base: Return token
            Base->>Base: Set Authorization header
            Base->>API: Proceed with request
        end
    end

    API->>AW: Make API call

    alt Success (200)
        AW-->>API: Return data
    else 401 Unauthorized
        AW-->>API: Token invalid
        API->>Base: Handle 401
        Base->>Base: Clear Authorization header
        Base->>Auth: Get fresh token
        Auth->>AW: POST /api/v1/authentication/login
        AW-->>Auth: Return new token
        Auth->>DB: Update token & expiry
        Base->>Base: Update Authorization header
        Base->>AW: Retry API call
        AW-->>API: Return data
    end
```

8.4 Key Functions

8.4.1 `ensure_authenticated_headers()`

Located in `base_api.py`:

```
def ensure_authenticated_headers(self):
    """Ensure headers have valid bearer token"""
    # Only get token if we don't already have one in headers
    if "Authorization" not in self.headers:
        token = self.get_valid_token()
        self.headers["Authorization"] = f"Bearer {token}"
```

Purpose: Adds Authorization header only if not already present.

When Called: Before every API request (GET, POST, PUT, DELETE)

8.4.2 `get_valid_token()`

```
def get_valid_token(self):
    """Get a valid bearer token, authenticate if needed"""
    # Check if we have a cached token that's still valid in the databa
    self.airwallex_setting.reload()
    cached_token = self.airwallex_setting.token
    expires_at = self.airwallex_setting.token_expiry

    if cached_token and expires_at:
        # Check if token is still valid (with 5 minute buffer)
        if frappe.utils.now_datetime() < frappe.utils.get_datetime(exp
            return cached_token

    # Token expired or doesn't exist, get a new one
    return self.authenticate_and_cache_token()
```

Purpose: Returns a valid token, fetching new one if needed.

Logic: 1. Reload settings from database (get latest token) 2. Check if token exists and is not expired (with 5-min buffer) 3. Return cached token if valid 4. Otherwise, authenticate and get new token

8.4.3 `authenticate_and_cache_token()`

```
def authenticate_and_cache_token(self):
    """Authenticate and cache the token in database"""
    auth = AirwallexAuthenticator()
    auth_response = auth.authenticate()

    if not auth_response or not auth_response.get('token'):
        frappe.throw("Failed to authenticate with Airwallex API")

    token = auth_response['token']
    expires_at = auth_response.get('expires_at')

    # Save token to database
    self.airwallex_setting.token = token
    if expires_at:
        if isinstance(expires_at, str):
            expires_datetime = datetime.fromisoformat(expires_at.repla
        else:
            expires_datetime = expires_at
        self.airwallex_setting.token_expiry = expires_datetime

    self.airwallex_setting.save(ignore_permissions=True)
    frappe.db.commit()

    return token
```

Purpose: Authenticate with Airwallex and cache the token.

Process: 1. Call Airwallex authentication endpoint 2. Extract token and expiry from response 3. Save to database (Bank Integration Setting) 4. Commit transaction 5. Return token

8.4.4 401 Error Handling

Located in `_make_request()`:

```
# Check for authentication errors (401 Unauthorized)
if response.status_code == 401 and not self.is_auth_instance:
    frappe.logger().info("Token expired, refreshing and retrying reque

    # Clear the authorization header to force token refresh
    if "Authorization" in self.headers:
        del self.headers["Authorization"]

    # Get new token and update headers
    token = self.authenticate_and_cache_token()
    self.headers["Authorization"] = f"Bearer {token}"
    request_headers["Authorization"] = f"Bearer {token}"

    # Retry the request with new token
    response = requests.request(method.value, url, params=params, jso
```

Purpose: Automatically recover from token expiration.

Process: 1. Detect 401 response 2. Clear existing Authorization header 3. Get fresh token 4. Update headers 5. Retry the original request 6. Return result (success or failure)

8.5 Token Lifecycle

8.5.1 1. First Request

- No token in database
- Authenticate with Airwallex
- Cache token with expiry
- Use token for request

8.5.2 2. Subsequent Requests

- Token exists in database
- Check if still valid (5-min buffer)
- Use cached token
- No authentication needed

8.5.3 3. Token Near Expiry

- Token expires in < 5 minutes
- Proactively authenticate
- Update cached token
- Use new token

8.5.4 4. Token Expired

- API returns 401
- Automatically authenticate
- Update cached token
- Retry original request

8.6 Multi-Client Support

Each Airwallex client can have its own token:

```
# Client-specific token fields in Airwallex Client child table
token = DF.Small Text
token_expiry = DF.Datetime
```

When initializing API:

```
api = FinancialTransactions(
    client_id=client.airwallex_client_id,
    api_key=client.get_password("airwallex_api_key"),
    api_url=settings.api_url
)
```

The API client manages tokens for that specific client.

8.7 Security Considerations

8.7.1 Token Storage

- Tokens stored in database (not in logs)
- API keys stored as Password fields (encrypted)
- Tokens automatically expire (limited lifetime)

8.7.2 Token Exposure

- Never logged or shown in UI
- Masked in error messages
- Not transmitted unnecessarily

8.7.3 Token Refresh

- Automatic refresh on expiry
- No manual intervention required
- Minimal downtime during refresh

8.8 Performance Optimization

8.8.1 Minimized Authentication Requests

- Token reused until expiry
- 5-minute buffer prevents last-minute failures
- Single authentication serves multiple requests

8.8.2 Database vs Cache

- Database ensures consistency across workers
- Slight overhead vs in-memory cache
- Trade-off for reliability and persistence

8.8.3 Concurrent Requests

- Multiple simultaneous requests use same token
- No redundant authentication
- Thread-safe token management

8.9 Troubleshooting

8.9.1 “Failed to authenticate”

Cause: Invalid credentials or API unavailable **Solution**: - Verify client ID and API key - Check API URL is correct - Test authentication via UI button

8.9.2 Token Keeps Expiring

Cause: System time incorrect or Airwallex token lifetime changed**Solution**: - Verify server time is accurate - Check token expiry in database - Contact Airwallex if lifetime changed

8.9.3 401 Errors Despite Fresh Token

Cause: Token invalidated server-side **Solution**: - Check Airwallex account status - Verify API key hasn’t been revoked - Re-enter credentials if needed

8.10 Best Practices

1. **Don’t Force Refresh**: Let the system manage tokens automatically
2. **Monitor Token Expiry**: Check logs for frequent re-authentication (may indicate issues)
3. **Update Credentials Carefully**: Changing credentials clears cached tokens
4. **Test After Changes** : Use test authentication after updating credentials
5. **Secure API Keys**: Treat API keys like passwords, never expose in logs or UI

9 Data Mapping

9.1 Overview

This document describes how Airwallex transaction data is mapped to ERPNext Bank Transaction format, including transaction filtering logic.

9.2 Core Functions

9.2.1 Transaction Mapping Function

The core mapping function is located in`bank_integration/airwallex/utils.py`:

```
def map_airwallex_to_erpnext(txn, bank_account)
```

9.2.2 Transaction Filtering Function

The filtering logic is located in `bank_integration/bank_integration/doctype/bank_integration_setting/bank_integration_setting.`

```
def should_sync_transaction(self, transaction_type)
```

9.3 Transaction Filtering Logic

9.3.1 Pre-Processing Filter

Before mapping Airwallex data to ERPNext format, each transaction goes through a filtering process:

```
# In sync_client_transactions()
transaction_type = txn.get('transaction_type', '').upper()

# Check transaction type filtering
if not settings.should_sync_transaction(transaction_type):
    frappe.logger().info(f"Transaction {transaction_id} type '{transa
    processed += 1
    skipped += 1
    continue
```

9.3.2 Filtering Strategies

9.3.2.1 1. No Filters (Default)

```
# If no filters configured
if not self.transaction_type_filters:
    return True  # Sync all transactions
```

9.3.2.2 2. Whitelist Approach (Include Filters)

```
# If any "Include" filters exist
has_include_filters = any(f.filter_action == "Include" for f in self.t

if has_include_filters:
    # Only sync explicitly included types
    for filter_rule in self.transaction_type_filters:
        if filter_rule.transaction_type == transaction_type and filter
            return True
    return False  # Not in include list
```

9.3.2.3 3. Blacklist Approach (Exclude Filters)

```
# If only "Exclude" filters exist
for filter_rule in self.transaction_type_filters:
    if filter_rule.transaction_type == transaction_type and filter_rul
        return False  # Explicitly excluded

return True  # Not in exclude list, so sync
```

9.3.3 Supported Transaction Types

The system supports all Airwallex transaction types:

```
DISPUTE_REVERSAL, DISPUTE_LOST, REFUND, REFUND_REVERSAL,
REFUND_FAILURE,
PAYMENT_RESERVE_HOLD, PAYMENT_RESERVE_RELEASE, PAYOUT,
PAYOUT_FAILURE,
PAYOUT_REVERSAL, CONVERSION_SELL, CONVERSION_BUY,
CONVERSION_REVERSAL,
DEPOSIT, ADJUSTMENT, FEE, DD_CREDIT, DD_DEBIT, DC_CREDIT, DC_DEBIT,
TRANSFER, PAYMENT, ISSUING_AUTHORISATION_HOLD,
ISSUING_AUTHORISATION_RELEASE,
ISSUING_CAPTURE, ISSUING_REFUND, PURCHASE, PREPAYMENT,
PREPAYMENT_RELEASE
```

9.4 Field Mapping

9.4.1 Direct Mappings

ERPNext Field	Airwallex Field	Transformation
`date`	`created_at`	Extract date part (YYYY-MM-DD)
`currency`	`currency`	Direct mapping
`description`	`description` or `source_type`	Use description, fallback to source_type
`reference_number`	`batch_id`	Direct mapping
`transaction_id`	`id`	Direct mapping (used for duplicate detection)
`transaction_type`	`transaction_type`	Direct mapping
`airwallex_source_type`	`source_type`	Custom field
`airwallex_source_id`	`source_id`	Custom field

9.4.2 Conditional Mappings

9.4.2.1 Status Mapping

```
def map_airwallex_status_to_erpnext(airwallex_status):
    status_mapping = {
        "PENDING": "Unreconciled",
        "SETTLED": "Settled",
        "CANCELLED": "Cancelled"
    }
    return status_mapping.get(airwallex_status.upper(), "Unreconciled"
```

Airwallex Status	ERPNext Status
PENDING	Unreconciled
SETTLED	Settled
CANCELLED	Cancelled
Other	Unreconciled (default)

9.4.2.2 Amount Mapping

```
amount = txn.get("net", 0)
is_deposit = amount > 0

# In ERPNext format:
"deposit": amount if is_deposit else 0,
"withdrawal": abs(amount) if not is_deposit else 0
```

- **Positive amount**: Mapped to `deposit`
- **Negative amount**: Mapped to `withdrawal` (absolute value)

9.4.2.3 Bank Account Mapping

```
# Get transaction currency
txn_currency = txn.get("currency", "")

# Check if bank account currency matches
mapped_bank_account = None
if bank_account and txn_currency:
    # Fetch the bank account's currency
    account = frappe.db.get_value("Bank Account", bank_account, "accou
    bank_account_currency = frappe.db.get_value("Account", account, "a

    # Only map if currencies match
    if bank_account_currency == txn_currency:
        mapped_bank_account = bank_account
    else:
        frappe.logger().info("Currency mismatch")
```

Logic: 1. Get transaction currency 2. Get bank account's linked GL account currency 3. Compare currencies 4. If match: assign bank account 5. If mismatch: leave bank account blank

Rationale: Prevents incorrectly assigning transactions to wrong-currency bank accounts.

9.5 Complete Processing Flow

9.5.1 1. Transaction Retrieval

```
# Get transactions from Airwallex API
transactions = api.get_list(from_created_at=from_date_iso, to_created_
```

9.5.2 2. Pre-Processing Validation

```
# Check if transaction already exists (duplicate prevention)
if transaction_exists(transaction_id):
    continue

# Check transaction type filtering
if not settings.should_sync_transaction(transaction_type):
    continue

# Check basic currency validation
if not transaction_currency:
    continue
```

9.5.3 3. Data Mapping

```
# Map Airwallex transaction to ERPNext format
bank_txn = map_airwallex_to_erpnext(txn, client.bank_account)
```

9.5.4 4. Document Creation

```
# Create and submit ERPNext Bank Transaction
bank_txn_doc = frappe.get_doc(bank_txn)
bank_txn_doc.insert()
bank_txn_doc.submit()
```

9.6 Sample Transformation

9.6.1 Input (Airwallex)

```
{
  "amount": 200.21,
  "batch_id": "bat_20201202_SGD_2",
  "client_rate": 6.93,
  "created_at": "2021-03-22T16:08:02",
  "currency": "CNY",
  "description": "deposit to account",
  "id": "7f687fe6-dcf4-4462-92fa-80335301d9d2",
  "net": 100.21,
  "settled_at": "2021-03-22T16:08:02",
  "source_id": "9f687fe6-dcf4-4462-92fa-80335301d9d2",
  "source_type": "PAYMENT_ATTEMPT",
  "status": "PENDING",
  "transaction_type": "PAYMENT"
}
```

9.6.2 Transaction Filter Check

```
# Check if PAYMENT type should be synced
settings = frappe.get_single("Bank Integration Setting")
should_sync = settings.should_sync_transaction("PAYMENT")

# If filter exists and excludes PAYMENT, or no include filter for PAYM
if not should_sync:
    # Transaction is skipped, not mapped
    return
```

9.6.3 Output (ERPNext Bank Transaction)

```
{
  "doctype": "Bank Transaction",
  "date": "2021-03-22",
  "status": "Unreconciled",
  "bank_account": "CNY Bank Account",  # Only if currency matches
  "currency": "CNY",
  "description": "deposit to account",
  "reference_number": "bat_20201202_SGD_2",
  "transaction_id": "7f687fe6-dcf4-4462-92fa-80335301d9d2",
  "transaction_type": "PAYMENT",
  "deposit": 100.21,
  "withdrawal": 0,
  "airwallex_source_type": "PAYMENT_ATTEMPT",
  "airwallex_source_id": "9f687fe6-dcf4-4462-92fa-80335301d9d2"
}
```

9.7 Error Handling

9.7.1 Filtering Errors

- **Unknown transaction type**: Filtered based on default strategy
- **Missing transaction type**: Skipped with warning log
- **Filter configuration errors**: Logged but don't stop sync

9.7.2 Mapping Errors

- **Currency mismatch**: Bank account left blank, transaction still created
- **Missing required fields**: Transaction skipped with error log
- **Amount parsing errors**: Default to 0, log warning

9.7.3 Duplicate Handling

- **Existing transaction_id**: Skip mapping, increment skip counter
- **Database constraint violations**: Catch and skip gracefully

9.8 Performance Considerations

9.8.1 Filtering Impact

- **Early filtering**: Reduces unnecessary mapping operations
- **Database queries**: Currency check requires 2 DB queries per transaction
- **Logging**: Filtered transactions are logged for audit trail

9.8.2 Optimization Techniques

- **Batch processing**: Process transactions in chunks
- **Progress updates**: Update progress every 10 transactions
- **Connection pooling**: Reuse database connections
- **Token caching**: Avoid repeated authentication

9.9 Monitoring and Debugging

9.9.1 Log Messages

```
# Successful mapping
frappe.logger().info(f"Created transaction {transaction_id} of type {t

# Filtered transaction
frappe.logger().info(f"Transaction {transaction_id} type '{transactio

# Currency mismatch
frappe.logger().info(f"Currency mismatch: Transaction {transaction_id}
```

9.9.2 Counters

- `processed`: Total transactions examined
- `created`: Successfully mapped and created
- `skipped`: Filtered out or duplicates
- `errors`: Failed mapping attempts

9.9.3 Best Practices

1. **Monitor skip ratios**: High skip rates may indicate filter misconfiguration
2. **Check currency mismatches**: May indicate wrong bank account assignment
3. **Review error logs**: Failed mappings indicate data issues
4. **Test filters thoroughly**: Use small date ranges to verify behavior

10 Error Handling & Recovery

10.1 Overview

The Bank Integration App implements comprehensive error handling at multiple levels to ensure robust operation and easy troubleshooting.

10.2 Error Handling Layers

10.2.1 1. Configuration Validation

Location: `bank_integration_setting.py` - `validate()` method

Checks: - At least one Airwallex client configured - Required fields (Client ID, API Key, Bank Account) populated - Authentication succeeds before enabling integration

Behavior:

```
def validate(self):
    if self.enable_airwallex:
        if self._credentials_changed():
            if not self.test_authentication_silent():
                self.enable_airwallex = 0
                frappe.msgprint("   Authentication failed", indicator
```

User Experience: Integration auto-disabled on validation failure.

10.2.2 2. API-Level Errors

Location: `base_api.py` - `_make_request()` method

10.2.2.1 Authentication Errors (401)

```
if response.status_code == 401 and not self.is_auth_instance:
    # Clear auth header
    # Get fresh token
    # Retry request automatically
```

Handling: Automatic retry with fresh token

10.2.2.2 HTTP Errors (4xx, 5xx)

```
if response.status_code >= 400:
    error_msg = f"HTTP {response.status_code}: {response.text}"
    frappe.throw(_("Airwallex API request failed: {0}").format(error_r
```

Handling: Throw error with detailed message

10.2.2.3 Network Errors

```
except Exception as e:
    error_response = response.text if response else str(e)
    self.create_connection_log(status=500, ...)
    frappe.throw(_("Airwallex API request failed: {0}").format(str(e)
```

Handling: Log error and throw exception

10.2.3 3. Transaction-Level Errors

Location: `transaction.py` - `sync_client_transactions()` function

10.2.3.1 Duplicate Transactions

```
if transaction_exists(transaction_id):
    total_skipped += 1
    continue
```

Handling: Skip silently, increment counter

10.2.3.2 Duplicate Entry Errors

```
except frappe.DuplicateEntryError:
    total_processed += 1
    total_skipped += 1
    existing_transaction_ids.add(txn.get('id'))
```

Handling: Catch database constraint violation, skip transaction

10.2.3.3 Mapping Errors

```
if not erpnext_txn.get('transaction_id'):
    frappe.logger().warning("Transaction missing transaction_id")
    total_errors += 1
    total_processed += 1
    continue
```

Handling: Log warning, increment error counter, continue

10.2.3.4 General Transaction Errors

```
except Exception as e:
    total_errors += 1
    total_processed += 1
    frappe.logger().error(f"Error processing transaction: {str(e)}")
    frappe.log_error(traceback.format_exc(), f"Transaction Sync Error"
```

Handling: Log error, continue with next transaction

10.2.4 4. Client-Level Errors

Location: `transaction.py` - `sync_transactions()` function

```
for client in settings.airwallex_clients:
    try:
        processed, created = sync_client_transactions(...)
        total_processed += processed
        total_created += created
    except Exception as e:
        client_short = client.airwallex_client_id[:8]
        frappe.log_error(
            message=f"Failed to sync for client {client.airwallex_cli
            title=f"Sync Error - {client_short}"
        )
```

Handling: Log error, continue with next client

10.2.5 5. Sync-Level Errors

Location: `transaction.py` - `sync_scheduled_transactions()` function

```
try:
    sync_transactions(start_date, end_date, setting_name)
    setting.db_set('last_sync_date', frappe.utils.now())
except Exception as e:
    try:
        setting.db_set('sync_status', 'Failed')
    except:
        pass
    frappe.log_error(frappe.get_traceback(), f"Scheduled Sync Error")
```

Handling: Set status to Failed, log error, preserve state

10.3 Error Logging

10.3.1 Bank Integration Log

Custom logging via `bank_integration_log.py`:

```
bi_log.create_log(
    message="Starting sync from X to Y",
    status="Info"  # or "Error"
)
```

Purpose: User-friendly sync history

10.3.2 Frappe Error Log

Standard error logging:

```
frappe.log_error(
    message="Detailed error message",
    title="Short Error Title"
)
```

Purpose: Developer-level debugging

10.3.3 Application Logger

```
frappe.logger().info("Info message")
frappe.logger().warning("Warning message")
frappe.logger().error("Error message")
```

Purpose: Console/file logging for monitoring

10.4 Error Recovery Strategies

10.4.1 Automatic Recovery

Error Type	Recovery Method
401 Unauthorized	Auto-refresh token and retry
Duplicate transaction	Skip and continue
Single transaction error	Log and continue with next
Network timeout	Fail and log (retry next scheduled run)

10.4.2 Manual Recovery

Error Scenario	Recovery Steps
Authentication failure	1. Check credentials 2. Click “Test Authentication” 3. Re-enable if successful
Sync stuck “In Progress”	1. Wait for timeout 2. Click “Restart Sync”
Missing transactions	1. Note date range 2. Use manual sync to backfill
Currency mismatch	1. Review bank account configuration 2. Ensure currencies match 3. Re-sync affected period

10.5 Concurrent Sync Prevention

```
if setting.sync_status == "In Progress":
    frappe.logger().info("Sync already in progress, skipping")
    return
```

Purpose: Prevent multiple simultaneous syncs that could cause: - Duplicate transactions - Database locks - Resource contention

10.6 Error Notification

10.6.1 Real-time Notifications

```
frappe.publish_realtime(
    'transaction_sync_complete',
    {
        'processed': total_processed,
        'created': total_created,
        'skipped': total_skipped,
        'errors': total_errors,
        'status': final_status,
        'message': "Summary message"
    },
    user=frappe.session.user
)
```

Delivery: Immediately to active user session

10.6.2 Status Updates

```
def update_sync_progress(self, processed, total, status="In Progress")
    self.db_set('sync_status', status)
    self.db_set('processed_records', processed)
    self.db_set('total_records', total)
    # ... publish realtime update
```

Purpose: Keep UI informed of current state

10.7 Common Error Messages

10.7.1 “No Airwallex clients configured”

Cause: No clients in Airwallex Clients table **Fix**: Add at least one client with valid credentials

10.7.2 “Authentication failed for one or more clients”

Cause: Invalid Client ID or API Key **Fix**: Verify credentials, test authentication

10.7.3 “From and To dates are required”

Cause: Missing dates for manual sync **Fix**: Set both From Date and To Date

10.7.4 “Sync already in progress”

Cause: Another sync is running **Fix**: Wait for completion or reset status

10.7.5 “Transaction sync failed: [error details]”

Cause: Various (network, API, data issues) **Fix**: Check Error Log for details, address root cause

10.8 Debugging Tools

10.8.1 Check Sync Status

```
frappe.get_single("Bank Integration Setting").sync_status
```

10.8.2 View Recent Errors

```
SELECT * FROM `tabError Log`
WHERE creation > DATE_SUB(NOW(), INTERVAL 1 DAY)
AND error LIKE '%airwallex%'
ORDER BY creation DESC;
```

10.8.3 Count Synced Transactions

```
SELECT COUNT(*) FROM `tabBank Transaction`
WHERE transaction_id IS NOT NULL
AND transaction_id != '';
```

10.8.4 Find Duplicates

```
SELECT transaction_id, COUNT(*) as count
FROM `tabBank Transaction`
GROUP BY transaction_id
HAVING count > 1;
```

10.9 Best Practices

1. **Monitor Error Logs**: Review regularly for patterns
2. **Test Credentials**: After any configuration change
3. **Start Small**: Test with short date ranges first
4. **Review Logs**: Check Bank Integration Log after each sync
5. **Handle Errors Gracefully**: Don’t panic - most errors are recoverable
6. **Document Issues**: Note error patterns for future reference
7. **Update Regularly**: Keep app updated for bug fixes

11 Transaction Mapping Reference

ERPNext Field	Airwallex Field	Description
`date`	`created_at`	Transaction creation timestamp (convert to YYYY-MM-DD)
`status`	`status`	Transaction status (`PENDING`, `SETTLED`, `CANCELLED`), equivalent to `Pending`, `Settled`, `Unreconciled`, `Reconciled`, `Cancelled` in ERPNext
`bank_account`	`funding_source_id`	Map to ERPNext Bank Account via lookup
`currency`	`currency`	ISO 4217 currency code (e.g. CNY, USD)
`description`	`description` or `source_type`	Use `description` if available, fallback to `source_type`
`reference_number`	`batch_id`	Optional batch reference ID
`transaction_id`	`id`	Unique Airwallex transaction ID
`transaction_type`	`transaction_type`	e.g. `PAYMENT`, `CONVERSION`, etc. (see transaction types below)
`deposit`	`net` (if positive)	Net amount received (used for incoming funds)
`withdrawal`	`net` (if negative)	Net amount paid out (used for outgoing funds)
`bank_party_name`	-	Standard ERPNext field (not mapped from Airwallex)
`bank_party_account_number`	-	Standard ERPNext field (not mapped from Airwallex)
Custom Field: `airwallex_source_type`	`source_type`	Transaction source type (see options below)
Custom Field: `airwallex_source_id`	`source_id`	Optional: source identifier from Airwallex

11.1 Custom Fields Required

The following custom fields need to be added to the ERPNext Bank Transaction doctype:

1. **`airwallex_source_type`** (Data field)
   - Label: "Airwallex Source Type"
   - Field Type: Data
   - Options: None (stores raw Airwallex source_type values)
2. **`airwallex_source_id`** (Data field)
   - Label: "Airwallex Source ID"
   - Field Type: Data
   - Description: "Source identifier from Airwallex transaction"

11.2 Airwallex Source Type Options

The `source_type` field can contain the following values:

- `CONVERSION` - Currency conversion transactions
- `DEPOSIT` - Incoming deposit transactions
- `ADJUSTMENT` - Account balance adjustments
- `FEE` - Platform or service fees
- `PAYMENT_ATTEMPT` - Payment processing attempts
- `REFUND` - Refund transactions
- `DISPUTE` - Chargeback or dispute transactions
- `CHARGE` - Direct charges or debits
- `TRANSFER` - Transfer transactions between accounts
- `YIELD` - Interest or yield payments
- `BATCH_PAYOUT` - Batch payout transactions
- `CARD_PURCHASE` - Card-based purchase transactions
- `CARD_REFUND` - Card-based refund transactions
- `PURCHASE` - General purchase transactions
- `REFUND_REVERSAL` - Reversed refund transactions

11.3 Airwallex Transaction Type Options

The `transaction_type` field can contain the following values:

- `DISPUTE_REVERSAL` - Reversal of dispute transactions
- `DISPUTE_LOST` - Lost dispute transactions
- `REFUND` - Refund transactions
- `REFUND_REVERSAL` - Reversed refund transactions
- `REFUND_FAILURE` - Failed refund transactions
- `PAYMENT_RESERVE_HOLD` - Payment reserve hold transactions
- `PAYMENT_RESERVE_RELEASE` - Payment reserve release transactions
- `PAYOUT` - Payout transactions
- `PAYOUT_FAILURE` - Failed payout transactions
- `PAYOUT_REVERSAL` - Reversed payout transactions
- `CONVERSION_SELL` - Currency conversion sell transactions
- `CONVERSION_BUY` - Currency conversion buy transactions
- `CONVERSION_REVERSAL` - Reversed conversion transactions
- `DEPOSIT` - Deposit transactions
- `ADJUSTMENT` - Account adjustment transactions
- `FEE` - Fee transactions
- `DD_CREDIT` - Direct debit credit transactions
- `DD_DEBIT` - Direct debit debit transactions
- `DC_CREDIT` - Direct credit transactions
- `DC_DEBIT` - Direct debit transactions
- `TRANSFER` - Transfer transactions
- `PAYMENT` - Payment transactions
- `ISSUING_AUTHORISATION_HOLD` - Card issuing authorization hold
- `ISSUING_AUTHORISATION_RELEASE` - Card issuing authorization release
- `ISSUING_CAPTURE` - Card issuing capture transactions
- `ISSUING_REFUND` - Card issuing refund transactions
- `PURCHASE` - Purchase transactions
- `PREPAYMENT` - Prepayment transactions
- `PREPAYMENT_RELEASE` - Prepayment release transactions