Step-by-Step AccuFund Migration Guide (v9.0)

This comprehensive guide provides detailed, step-by-step instructions for migrating your financial data from AccuFund to the Nonprofit Fund Accounting System. Follow these instructions carefully to ensure a successful migration with minimal disruption to your operations.

> **Note:** This guide is updated for version 9.0 of the Nonprofit Fund Accounting System, which includes enhanced support for *Inter-Entity Transfers* (v9.0) **and the Bank Account Management module** (v9.0+). If you're migrating to an earlier version, some steps may not apply.

Table of Contents

1. Preparation and Assessment

1 Create a Migration Project Plan

Begin by creating a detailed project plan with timeline, resources, and responsibilities.

1. Form a migration team with representatives from finance, IT, and management
2. Establish a timeline with key milestones and deadlines
3. Schedule the migration during a low-activity period (e.g., after month-end closing)
4. Create a communication plan to keep stakeholders informed

2 Assess Your AccuFund Implementation

Document your current AccuFund setup to understand what needs to be migrated.

- Document your chart of accounts structure and hierarchy
- List all funds and their relationships
- Identify all entities in your organization structure
- Document any customizations or special configurations
- List all reports you regularly use
- Identify any third-party integrations

> **Tip:** Use the AccuFund System Administrator module to generate reports on your configuration.

---

3 Set Up the Migration Environment

Prepare the technical environment for the migration process.

1. Install PostgreSQL if not already installed:

```
# For macOS with Homebrew
brew install postgresql

# For Ubuntu/Debian
sudo apt update
sudo apt install postgresql postgresql-contrib
```

2. Create the fund_accounting_db database:

```
createdb -U postgres fund_accounting_db
```

3. Clone the Nonprofit Fund Accounting repository:

```
git clone https://github.com/tpfbill/nonprofit-fund-accounting.git
cd nonprofit-fund-accounting
git checkout v9.0
```

4. Install Node.js dependencies:

```
npm install
```

4 Create a Backup of Your AccuFund Data

Before proceeding, ensure you have a complete backup of your AccuFund database.

1. Use AccuFund's backup utility to create a full system backup
2. Verify the backup is complete and can be restored if needed
3. Store the backup in a secure location

> **Warning:** Never proceed with migration without a verified backup. The migration process is irreversible once data is committed to the new system.

2. Data Extraction from AccuFund

1 Extract Chart of Accounts

Export your complete chart of accounts from AccuFund.

1. In AccuFund, go to **General Ledger > Reports > Chart of Accounts**
2. Select the option to include all fields
3. Choose CSV as the export format
4. Save the file as `accufund_coa.csv`

Ensure the export includes these critical fields:

- Account Code
- Account Name
- Account Type
- Account Status
- Parent Account (if hierarchical)
- Entity/Department

2 Extract Fund Definitions

Export all fund definitions from AccuFund.

1. In AccuFund, go to **General Ledger > Reports > Fund Listing**
2. Select the option to include all fields
3. Choose CSV as the export format
4. Save the file as `accufund_funds.csv`

---

3 Extract Journal Entries

Export all journal entries for the migration period.

1. In AccuFund, go to **General Ledger > Reports > Journal Entry Detail**
2. Set the date range to cover your migration period (typically current fiscal year plus prior year)
3. Select the option to include all fields
4. Choose CSV as the export format
5. Save the file as `accufund_journal_entries.csv`

> **Tip:** If you have a large volume of transactions, consider exporting in smaller batches by date range or fund.

---

4 Extract Entity Definitions

If you use multiple entities in AccuFund, export their definitions.

1. In AccuFund, go to **Administration > Organization Structure**
2. Export the organization hierarchy to CSV
3. Save the file as `accufund_entities.csv`

---

5 Extract Bank Account Information

Export all bank account information from AccuFund's Banking module.

1. In AccuFund, go to **Administration > Banking > Bank Accounts**
2. Generate a Bank Accounts Listing report

3. Include all fields, especially: Bank Name, Account Name, Account Number, Routing Number, Account Type, Status, and Current Balance
4. Export to CSV format
5. Save the file as `accufund_bank_accounts.csv`

> **Note:** Be sure to capture the Last Reconciled Date for each account to maintain reconciliation history.

---

6 Extract Balances

Export current account balances for verification purposes.

1. In AccuFund, go to **General Ledger > Reports > Trial Balance**
2. Set the date to the end of your last closed period
3. Export to CSV format
4. Save the file as `accufund_trial_balance.csv`

3. Data Mapping and Transformation

1 Create Entity Mapping

Map AccuFund entities to the new system's entity structure.

1. Create a spreadsheet with columns for AccuFund Entity ID, AccuFund Entity Name, New Entity ID, and New Entity Name
2. For each AccuFund entity, determine the corresponding entity in the new system
3. Save this mapping as `entity_mapping.csv`

Example mapping table:

AccuFund Entity ID	AccuFund Entity Name	New Entity ID	New Entity Name
MAIN	Main Organization	TPF_PARENT	The Principle Foundation

EDU	Education Division	TPF-ES	TPF Educational Services

---

2 Create Account Mapping

Map AccuFund accounts to the new system's chart of accounts.

1. Create a spreadsheet with columns for AccuFund Account Code, AccuFund Account Name, New Account Code, New Account Name, and Account Type
2. For each AccuFund account, determine the corresponding account in the new system
3. Save this mapping as `account_mapping.csv`

> **Note:** Pay special attention to account types, as they might be categorized differently in the new system.

---

3 Create Fund Mapping

Map AccuFund funds to the new system's fund structure.

1. Create a spreadsheet with columns for AccuFund Fund Code, AccuFund Fund Name, New Fund Code, New Fund Name, and Fund Type
2. For each AccuFund fund, determine the corresponding fund in the new system
3. Save this mapping as `fund_mapping.csv`

---

4 Create Bank Account Mapping

Map AccuFund bank accounts to the new system's bank_accounts table structure.

1. Create a spreadsheet with the following columns to match the v9.0+ bank_accounts table:
    - bank_name
    - account_name
    - account_number
    - routing_number
    - type (map to: "Checking", "Savings", "Credit Card", etc.)

- status ("Active" or "Inactive")
- balance (current balance as of migration date)
- connection_method (default to "Manual" for initial import)
- description

2. For each AccuFund bank account, map the information to the corresponding fields
3. Save this mapping as `bank_account_mapping.csv`

> **Tip:** Include the GL account code that corresponds to each bank account to ensure proper reconciliation with the chart of accounts.

---

5 Transform Journal Entry Data

Process the journal entry data to match the new system's format.

1. Create a script to process the journal entries CSV file
2. Apply the entity, account, and fund mappings to transform the data
3. Format dates to match the new system's requirements (YYYY-MM-DD)
4. Split the data into two files:
    - `journal_entries.csv` - Header information for each journal entry
    - `journal_entry_lines.csv` - Line items for each journal entry

```
# Example Python script for transforming journal entries
import pandas as pd
import uuid

# Load mappings
entity_mapping = pd.read_csv('entity_mapping.csv')
account_mapping = pd.read_csv('account_mapping.csv')
fund_mapping = pd.read_csv('fund_mapping.csv')

# Load journal entries
journal_data = pd.read_csv('accufund_journal_entries.csv')

# Apply mappings and transformations
# [Transformation logic here]

# Save transformed data
```

```
journal_entries.to_csv('journal_entries.csv', index=False)
journal_entry_lines.to_csv('journal_entry_lines.csv', index=False)
```

4. Inter-Entity Transfer Migration (v9.0)

1 Identify Inter-Entity Transactions

Identify all transactions in AccuFund that represent transfers between entities.

1. Review your journal entries for transactions that involve multiple entities
2. Look for accounts with names like "Due To" or "Due From"
3. Create a list of all inter-entity transaction pairs
4. Save this list as `inter_entity_transactions.csv`

> **Tip:** In AccuFund, you can often identify these by running a report filtered on inter-company accounts or by specific transaction types used for inter-entity transfers.

---

2 Map Due To/Due From Accounts

Ensure proper mapping of Due To/Due From accounts to support the inter-entity transfer feature.

1. Identify all Due To/Due From accounts in AccuFund
2. Create or identify corresponding accounts in the new system following the required structure:
    - Asset accounts (19xx series) for "Due From" relationships
    - Liability accounts (29xx series) for "Due To" relationships
3. Update your account mapping to include these specialized accounts

> **Note:** The v9.0 Inter-Entity Transfer feature requires specific account structures. Each entity should have:
>
> - One "Due From" account (Asset, 19xx) for each counter-party entity
> - One "Due To" account (Liability, 29xx) for each counter-party entity

Example naming convention:

Entity	Account Type	Account Code	Account Name
TPF	Asset	1901	Due From TPF-ES
TPF-ES	Liability	2901	Due To TPF

---

3 Transform Inter-Entity Transactions

Process inter-entity transactions to match the new system's format.

1. Create a script to identify and pair related inter-entity transactions
2. Generate a unique `matching_transaction_id` for each pair
3. Set the `is_inter_entity` flag to true for these transactions
4. Set the `target_entity_id` field to the appropriate entity
5. Update the transformed journal entries data with this information

```
# Example Python code for processing inter-entity transactions
import uuid

# For each pair of inter-entity transactions
for transaction_pair in inter_entity_pairs:
  # Generate a shared matching transaction ID
  matching_id = str(uuid.uuid4())

  # Update source transaction
  source_tx = journal_entries[journal_entries['id'] ==
transaction_pair['source_id']]
  source_tx['is_inter_entity'] = True
  source_tx['target_entity_id'] =
transaction_pair['target_entity_id']
  source_tx['matching_transaction_id'] = matching_id

  # Update target transaction
  target_tx = journal_entries[journal_entries['id'] ==
transaction_pair['target_id']]
  target_tx['is_inter_entity'] = True
```

```
  target_tx['target_entity_id'] =
transaction_pair['source_entity_id']
  target_tx['matching_transaction_id'] = matching_id
```

---

4 Handle Single-Entry Inter-Entity Transactions

If AccuFund recorded inter-entity transfers as single entries, split them into paired transactions.

1. Identify transactions that affect multiple entities but are recorded as single entries
2. Split each into two separate journal entries:
    - One for the source entity (with "Due From" account)
    - One for the target entity (with "Due To" account)
3. Link the pair with a common `matching_transaction_id`
4. Set the `is_inter_entity` flag to true for both entries
5. Set the appropriate `target_entity_id` for each entry

> **Warning:** This step is critical for maintaining proper inter-entity relationships. Single-entry transactions must be split to work with the v9.0 Inter-Entity Transfer feature.

5. Data Validation

1 Validate Chart of Accounts

Verify the transformed chart of accounts for completeness and accuracy.

1. Check that all accounts from AccuFund have been mapped
2. Verify that account types are correctly assigned
3. Ensure that parent-child relationships are preserved
4. Confirm that entity associations are correct

```
# Example validation query
SELECT COUNT(*) FROM accounts;
# Compare with the count from AccuFund
```

2 Validate Fund Structure

Verify the transformed fund structure for completeness and accuracy.

1. Check that all funds from AccuFund have been mapped
2. Verify that fund types are correctly assigned
3. Confirm that entity associations are correct

3 Validate Bank Account Data

Verify the transformed bank account data for completeness and accuracy.

1. Check that all bank accounts from AccuFund have been mapped
2. Verify that account types are correctly assigned (Checking, Savings, Credit Card, etc.)
3. Ensure that account numbers and routing numbers are correctly formatted
4. Confirm that opening balances match the balances in AccuFund as of the migration date

> **Tip:** Pay special attention to bank accounts that have pending reconciliations or outstanding transactions.

4 Validate Journal Entries

Verify the transformed journal entries for completeness and accuracy.

1. Check that the total number of journal entries matches AccuFund
2. Verify that debits equal credits for each journal entry
3. Confirm that dates, amounts, and descriptions are preserved
4. Check that account and fund references are correctly mapped

5 Validate Inter-Entity Relationships

Verify that inter-entity transactions are correctly configured.

1. Check that all inter-entity transactions have been identified and paired
2. Verify that each pair shares the same `matching_transaction_id`
3. Confirm that `is_inter_entity` flags are set correctly
4. Verify that `target_entity_id` fields are correctly cross-referenced
5. Ensure that Due To/Due From accounts are used appropriately

> **Tip:** Create a validation report that shows each pair of inter-entity transactions side by side to verify they balance correctly.

---

6 Validate Balances

Verify that account balances match between AccuFund and the transformed data.

1. Calculate ending balances for all accounts using the transformed journal entries
2. Compare these with the trial balance exported from AccuFund
3. Investigate and resolve any discrepancies

```
# Example SQL to calculate account balances
SELECT
  a.id,
  a.code,
  a.name,
  COALESCE(SUM(jel.debit_amount), 0) AS total_debits,
  COALESCE(SUM(jel.credit_amount), 0) AS total_credits,
  COALESCE(SUM(jel.debit_amount - jel.credit_amount), 0) AS
balance
FROM accounts a
LEFT JOIN journal_entry_lines jel ON jel.account_id = a.id
GROUP BY a.id, a.code, a.name
ORDER BY a.code;
```

6. Data Import Process

1 Initialize the Database Schema

Set up the database schema in the new system.

1. Start the application to initialize the database schema:

```
cd nonprofit-fund-accounting
npm start
```

2. Verify that the database tables have been created:

```
psql -U postgres -d fund_accounting_db -c "\dt"
```

2 Import Entities

Import the entity structure into the new system.

1. Prepare the entities data in the required format
2. Import the entities using the provided script:

```
node add-tpf-hierarchy.js
```

3. Verify the entities were imported correctly:

```
psql -U postgres -d fund_accounting_db -c "SELECT * FROM
entities ORDER BY code"
```

3 Import Chart of Accounts

Import the chart of accounts into the new system.

1. Use the AccuFund import utility to import the accounts:

```
# Open the import utility in your browser
open http://localhost:3000/accufund-import.html
```

2. Upload the transformed `accounts.csv` file
3. Map the columns as prompted
4. Validate and import the data
5. Verify the accounts were imported correctly:

```
psql -U postgres -d fund_accounting_db -c "SELECT COUNT(*)
FROM accounts"
```

---

4 Import Funds

Import the fund structure into the new system.

1. Use the AccuFund import utility to import the funds:

```
# Open the import utility in your browser
open http://localhost:3000/accufund-import.html
```

2. Upload the transformed `funds.csv` file
3. Map the columns as prompted
4. Validate and import the data
5. Verify the funds were imported correctly:

```
psql -U postgres -d fund_accounting_db -c "SELECT COUNT(*)
FROM funds"
```

---

5 Import Bank Accounts

Import the bank accounts into the new system.

1. Use the AccuFund import utility to import the bank accounts:

```
# Open the import utility in your browser
open http://localhost:3000/accufund-import.html
```

2. Select "Bank Accounts" as the target table
3. Upload the transformed `bank_account_mapping.csv` file
4. Map the columns as prompted, ensuring all required fields are mapped correctly

5. Validate and import the data
6. Verify the bank accounts were imported correctly:

```
psql -U postgres -d fund_accounting_db -c "SELECT * FROM
bank_accounts ORDER BY bank_name, account_name"
```

> **Note:** The Bank Account Management module in v9.0+ requires proper setup of bank accounts to enable reconciliation and transaction matching features.

---

6 Import Journal Entries

Import the journal entries into the new system.

1. Import the journal entry headers:

```
# Using the COPY command for bulk import
psql -U postgres -d fund_accounting_db -c "\COPY
journal_entries FROM 'journal_entries.csv' WITH CSV HEADER"
```

2. Import the journal entry lines:

```
psql -U postgres -d fund_accounting_db -c "\COPY
journal_entry_lines FROM 'journal_entry_lines.csv' WITH CSV
HEADER"
```

3. Verify the journal entries were imported correctly:

```
psql -U postgres -d fund_accounting_db -c "SELECT COUNT(*)
FROM journal_entries"
psql -U postgres -d fund_accounting_db -c "SELECT COUNT(*)
FROM journal_entry_lines"
```

> **Warning:** Journal entry import can be time-consuming for large datasets. Consider importing in batches if you have a very large number of transactions.

7. Post-Migration Verification

1 Verify Entity Structure

Confirm that the entity structure has been correctly imported.

1. Navigate to **Settings > Entities** in the application
2. Verify that all entities are present and correctly organized
3. Check parent-child relationships

---

2 Verify Chart of Accounts

Confirm that the chart of accounts has been correctly imported.

1. Navigate to **Chart of Accounts** in the application
2. Verify that all accounts are present and correctly categorized
3. Check account types and entity associations

---

3 Verify Fund Structure

Confirm that the fund structure has been correctly imported.

1. Navigate to **Funds** in the application
2. Verify that all funds are present and correctly categorized
3. Check fund types and entity associations

---

4 Verify Bank Accounts

Confirm that the bank accounts have been correctly imported.

1. Navigate to **Bank Accounts** in the application
2. Verify that all bank accounts are present with correct information
3. Check account types, balances, and status
4. Perform a test reconciliation on one bank account to ensure the feature works properly
5. Verify that each bank account is properly linked to its corresponding GL account

> **Tip:** The Bank Account Management module added in v9.0 provides a comprehensive reconciliation workflow. Test this feature with a sample bank statement to ensure it works as expected.

---

5 Verify Financial Reports

Generate key financial reports and compare with AccuFund.

1. Generate a Trial Balance report in both systems
2. Generate a Balance Sheet report in both systems
3. Generate an Income Statement report in both systems
4. Compare the reports and investigate any discrepancies

---

6 Verify Inter-Entity Transfers

Confirm that inter-entity transfers have been correctly imported and are functioning as expected.

1. Navigate to **Reports** and run the Inter-Entity Transfers report
2. Verify that all expected transfer pairs are present and correctly linked
3. Check that Due To and Due From balances match between entities
4. Test the Inter-Entity Transfer Wizard to create a new transfer:
    - Navigate to **Inter-Entity Transfer** in the main menu
    - Create a test transfer between two entities
    - Verify that both journal entries are created correctly
    - Verify that the entries are linked with a matching transaction ID

> **Tip:** Run the following query to verify that inter-entity transfers are balanced:

```
SELECT
  e1.name AS source_entity,
  e2.name AS target_entity,
  je1.matching_transaction_id,
  je1.total_amount AS source_amount,
  je2.total_amount AS target_amount
FROM journal_entries je1
```

```
JOIN journal_entries je2 ON je1.matching_transaction_id =
je2.matching_transaction_id AND je1.id != je2.id
JOIN entities e1 ON je1.entity_id = e1.id
JOIN entities e2 ON je2.entity_id = e2.id
WHERE je1.is_inter_entity = true
ORDER BY je1.entry_date;
```

7 User Acceptance Testing

Have key users test the system to verify functionality.

1. Create a test plan covering key workflows
2. Have users execute the test plan and document results
3. Address any issues identified during testing
4. Get formal sign-off from key stakeholders

> **Note:** This is a critical step before going live with the new system. Ensure that all key users are involved in testing.

8. Troubleshooting Common Issues

1 Unbalanced Journal Entries

If journal entries are not balanced (debits don't equal credits):

1. Identify the unbalanced entries:

```
psql -U postgres -d fund_accounting_db -c "SELECT je.id,
je.reference_number, je.entry_date, SUM(jel.debit_amount) as
total_debits, SUM(jel.credit_amount) as total_credits,
SUM(jel.debit_amount) - SUM(jel.credit_amount) as difference
FROM journal_entries je JOIN journal_entry_lines jel ON
je.id = jel.journal_entry_id GROUP BY je.id,
je.reference_number, je.entry_date HAVING
```

```
ABS(SUM(jel.debit_amount) - SUM(jel.credit_amount)) > 0.01
ORDER BY je.entry_date"
```

2. Check the original AccuFund data for these entries
3. Fix the imbalances by adding missing lines or correcting amounts

---

2 Missing Accounts or Funds

If accounts or funds are missing after import:

1. Compare the counts between AccuFund and the new system
2. Identify the specific missing items
3. Check your mapping files for omissions
4. Import the missing items manually

---

3 Inter-Entity Transfer Issues

If inter-entity transfers are not working correctly:

1. Check that Due To/Due From accounts are properly set up with the correct naming and numbering conventions (19xx for assets, 29xx for liabilities)
2. Verify that each pair of transactions has the same `matching_transaction_id`
3. Confirm that `is_inter_entity` is set to true for both transactions
4. Ensure that `target_entity_id` is correctly cross-referenced
5. Verify that the entities exist and are correctly configured

```
# Fix missing matching_transaction_id
UPDATE journal_entries
SET matching_transaction_id = 'generated-uuid-here'
WHERE id IN ('entry1-id', 'entry2-id');
```

---

4 Balance Discrepancies

If account balances don't match between AccuFund and the new system:

1. Generate detailed transaction reports for the affected accounts in both systems

2. Compare the transactions line by line to identify differences
3. Check for missing transactions, incorrect amounts, or misclassifications
4. Make corrections as needed

---

5 Performance Issues

If the system is slow after migration:

1. Check database indexes:

```
psql -U postgres -d fund_accounting_db -c "\di"
```

2. Add missing indexes where needed:

```
psql -U postgres -d fund_accounting_db -c "CREATE INDEX
idx_journal_entries_entry_date ON
journal_entries(entry_date)"
```

3. Optimize database queries in the application code
4. Consider archiving older transactions if volume is very large

Conclusion

Following this step-by-step guide should result in a successful migration from AccuFund to the Nonprofit Fund Accounting System v9.0. The process requires careful planning, thorough validation, and attention to detail, especially for inter-entity transfers and bank account management.

After completing the migration, be sure to:

- Train all users on the new system
- Document any customizations or special configurations
- Establish regular backup procedures
- Keep the AccuFund data accessible for historical reference

For additional assistance, refer to the Administrator's Guide and User Guide for detailed information on using the system.

Generated by Droid by FACTORY on 7/22/2025