Jenzabar CX

General Ledger Reference

Technical Manual

Filename: tmglref
Distribution date: 08/18/1999

Contact us at www.jenzabar.com

JENZABAR, INC.
GENERAL LEDGER REFERENCE TECHNICAL MANUAL

TABLE OF CONTENTS

SECTION 1 - USING THIS MANUAL

Overview

Purpose of This Manual

This second volume of the CX General Ledger technical manual provides reference information. For specific information on implementing General Ledger, see Volume I of this manual.

Intended Audience

This guide is for use by those individuals responsible for the installation, customization, and maintenance of CX.

Product Differences

This manual contains documentation for all the features developed for the CX General Ledger module. Your institution may not have all these features.

Structure of This Manual

This manual is organized as follows:

SECTION 2 - GENERAL LEDGER PROCESSES

Overview

Introduction

This section provides information on the purpose and process flow of General Ledger.

Purpose of Module

The primary purpose of General Ledger is to enable the institution to perform accounting functions. Accounting functions include creating both routine and non-routine journal entries, reviewing the contents of the accounting records, and closing a fiscal period while preparing the accounting records for the next fiscal period.

Background Knowledge

The following list describes the necessary background information that you should know to implement and support the General Ledger module.

UNIX

Know the following about the UNIX operating system:

- Cash environment and commands
- Editor commands (e.g., vi)

INFORMIX-SQL

Know about the following INFORMIX tools:

- SQL database
- PERFORM screens
- ACE reports

Jenzabar CX database tools and utilities

Know how to use the following database tools:

- MAKE processor
- Schemas
- Macros
- Includes
- Program screens
- The SMO process

Jenzabar CX

Know the following about the CX standard product:

- CX directory structure
- The menu processor
- The CX database engine

QuickMate features

Know the following about the CX Graphical Server:

- Client/Server processing
- Network settings
- Keyboard settings
- Mouse settings
- GUI mode commands

C Programming

If you want to modify any CX programs to meet unique needs at your institution, you must know how to use the C programming language and have an in-depth knowledge of the CX code.

General Ledger policies and procedures

Know answers to the following questions:

- What are your institution’s accounting policies?
- What procedures does your institution use to create entries for payroll, accounts payable, fixed assets, and donor accounts?
- What are your institution’s policies for archiving obsolete accounting information?
- How does your institution use subsidiaries?
- Does your institution use claim on cash processing?
- What types of budgeting and reporting does your institution require? How do you use the account number structure to track your information needs?

Process Flows

Introduction

General Ledger consists of several different functions, including entry preparation and querying, and maintenance/year-end processing. This section contains a description of these functional aspects of General Ledger.

Entry Preparation and Querying

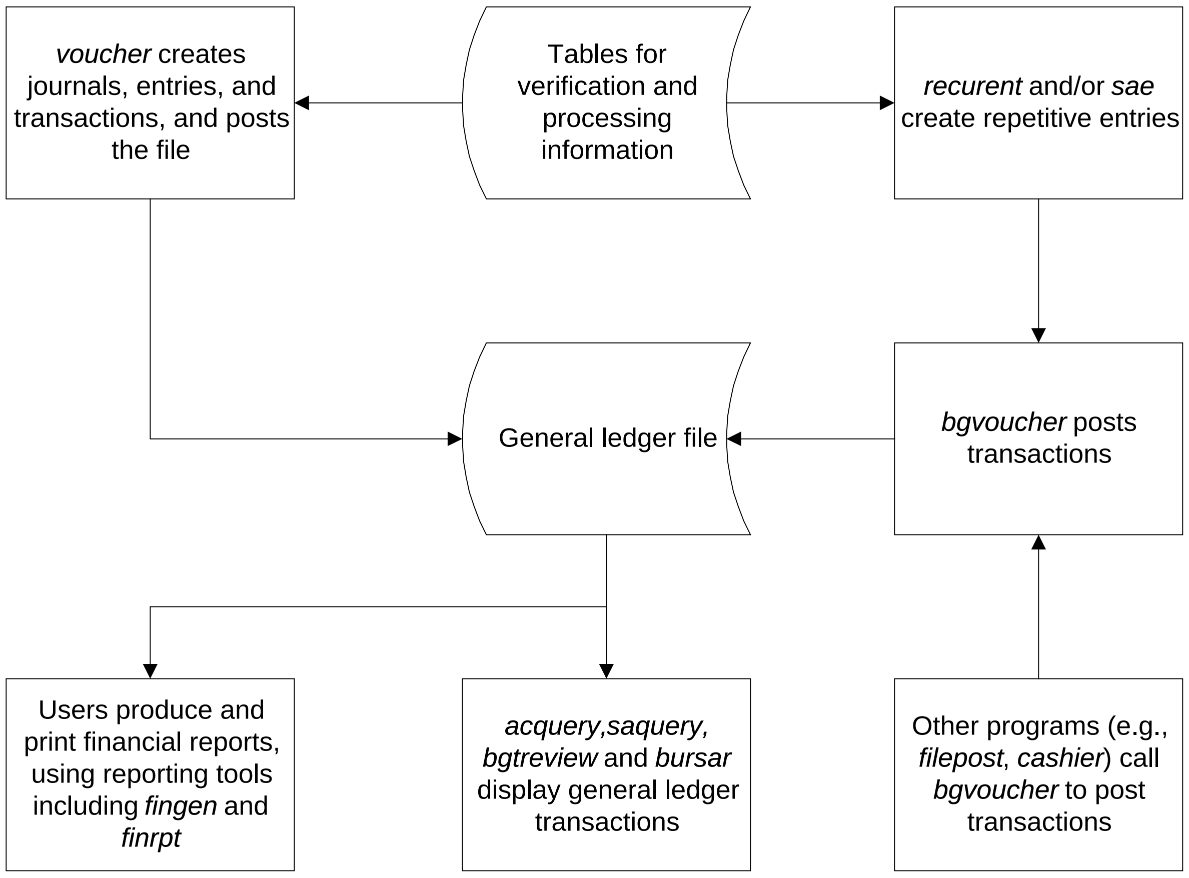
Users can create entries in a variety of ways using the General Ledger module. They can use *recurent* or *sae* to create repetitive entries, and they can create non-repetitive entries in *voucher*. *Voucher* and *bgvoucher* post the entries. In addition, other programs in CX call *filepost* to format a file for posting, and then *filepost* calls *bgvoucher* to post the formatted file.

After *voucher* and/or *bgvoucher* post the transactions, users can use the query programs *acquery* and *saquery* to view the contents of any accounting journals. *Bgtreview* enables users to view budget-related journals, and *bursar* provides the ability to view student account information.

Diagram

The following diagram shows the process flow for creating entries in General Ledger.

Note: For more information about program interrelationships and detailed data flow diagrams, see the individual program sections in this manual.



voucher creates journals, entries, and transactions, and posts the file
Tables for verification and processing information
recurent and/or *sae* create repetitive entries
General ledger file
bgvoucher posts transactions
Users produce and print financial reports, using reporting tools including *fingen* and *finrpt*
acquery,saquery, bgtreview and *bursar* display general ledger transactions
Other programs (e.g., *filepost*, *cashier*) call *bgvoucher* to post transactions

Maintenance/Year-end Processing

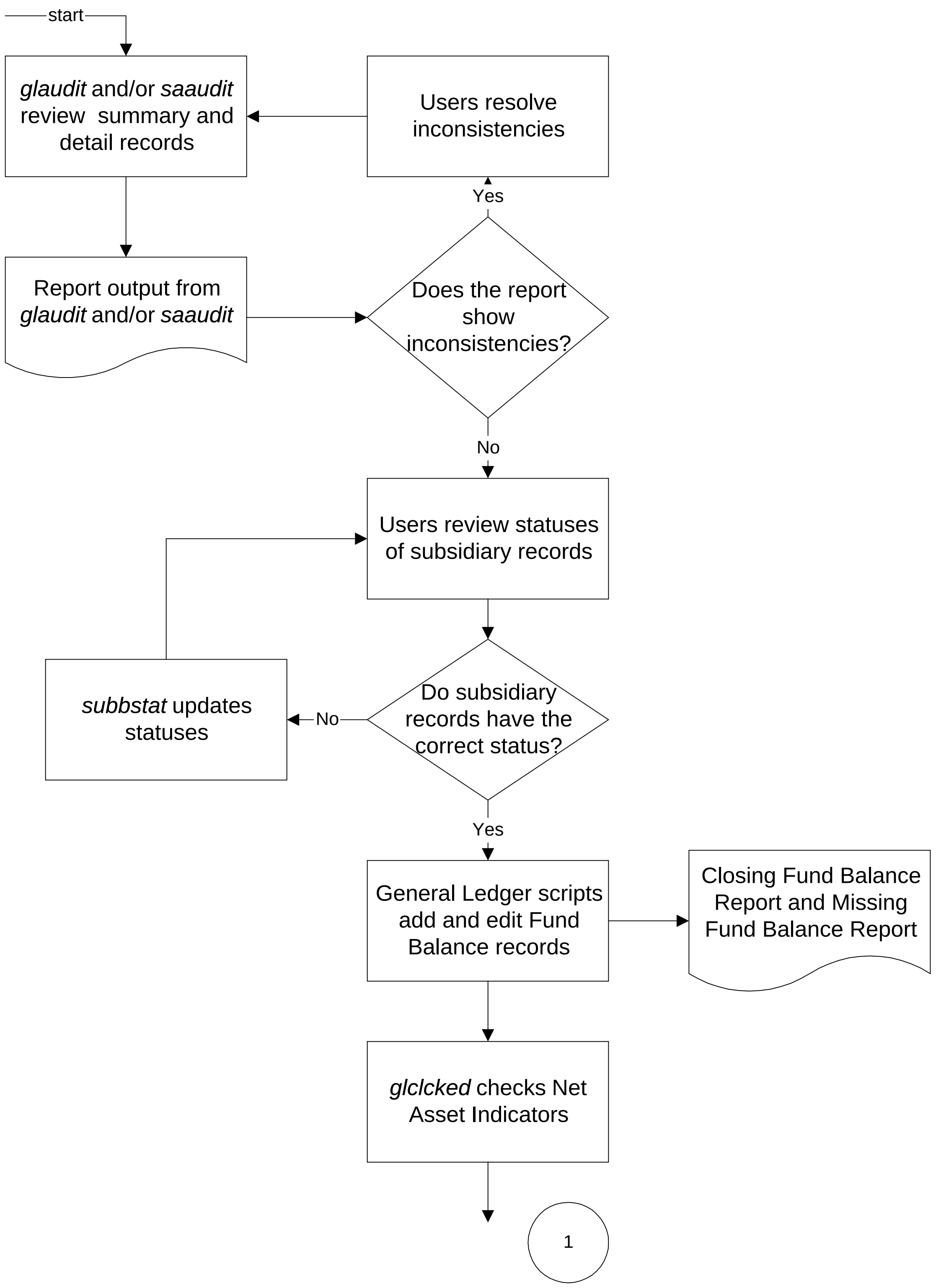
Maintenance and year-end processing consists of the following activities:

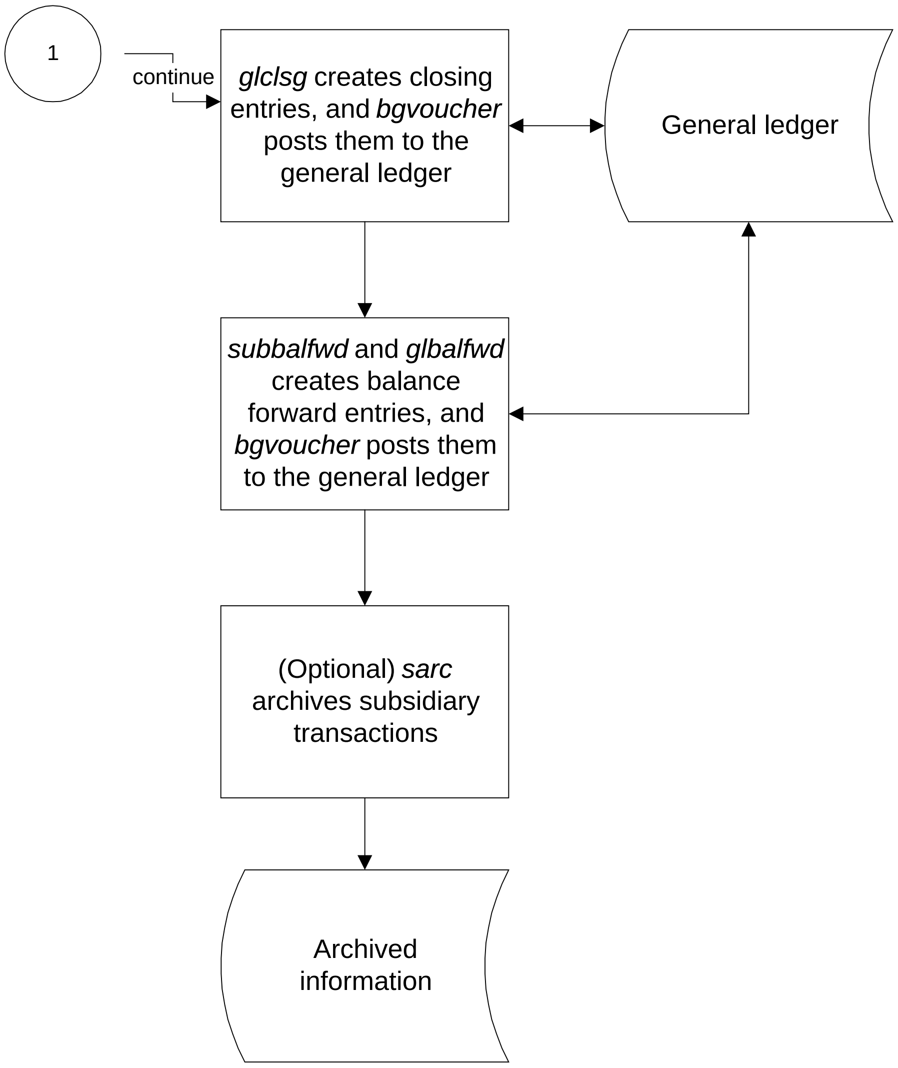
- Recovering files that were in use during a system failure
- Updating the status of subsidiary records
- Ensuring that summary records and detail records maintain consistent information
- Preparing balances for a new fiscal period

Diagram

The following diagram lays out the process flow for year-end processing in General Ledger.

Note: For more information about program interrelationships and detailed data flow diagrams, see the individual program sections in this manual.


start
glaudit and/or *saaudit* review summary and detail records
Users resolve inconsistencies
Yes
Report output from *glaudit* and/or *saaudit*
Does the report show inconsistencies?
No
Users review statuses of subsidiary records
subbstat updates statuses
No
Do subsidiary records have the correct status?
Yes
General Ledger scripts add and edit Fund Balance records
Closing Fund Balance Report and Missing Fund Balance Report
glclcked checks Net Asset Indicators
1


1
continue
glclsg creates closing entries, and *bgvoucher* posts them to the general ledger
General ledger
subbalfwd and *glbalfwd* creates balance forward entries, and *bgvoucher* posts them to the general ledger
(Optional) *sarc* archives subsidiary transactions
Archived information

Module Relationships

Other Jenzabar CX Modules

The General Ledger module interacts with several other modules in Jenzabar CX . The following list describes the interrelationships.

Note: Each of the programs that creates accounting entries uses the accounting program *bgvoucher* to post the transactions.

Alumni/Development
The Alumni/Development office processes information about gifts to the institution, and uses Donor Accounting to record gifts.

Personnel/Payroll
The Payroll department processes advances, paychecks, and direct deposits for employees, creating accounting transactions.

Accounts Payable
The Purchasing and Accounts Payable department writes checks in payment for goods and services received, creating accounting transactions.

Fixed Assets
The Fixed Assets module tracks acquisitions, disposals, and depreciation for long-lived assets.

Student Billing
The Student Billing module computes amounts due from students for tuition and other services, creating accounting transactions.

Fee Collection
The Fee Collection module allows you to collect payments and allocate them to the proper fees.

Cashier
The Cashier module allows you to perform accounting functions associated with cash receipts and disbursements.

SECTION 3 - GENERAL LEDGER TABLES AND RECORDS

Overview

Introduction

This section provides you with reference information about each table and record associated with the General Ledger module. The following tables appear in this section:

- Amount Type table (atype_table)
- Claim table (claim_table)
- Combined Centers table (cntrcomb_table)
- Document table (doc_table)
- Entry table (ent_table)
- Financial Statement table (fs_table)
- Financial table (fin_rpt_table)
- Function table (func_table)
- Fund table (fund_table)
- General Ledger Association table (glas_table)
- Journal Type table (vch_table)
- Object table (obj_table)
- Recur table (recur_table)
- Standard Accounting Entries table (sae_table)
- Subsidiary Association table (subas_table)
- Subsidiary Balance table (subb_table)
- Subsidiary table (subs_table)
- Subsidiary Total table (subt_table)

The following records appear in this section:

- Closing Fund Balance record (clsgfb_rec)
- Defined Account record (gld_rec)
- Financial Format record (fin_fmt_rec)
- Financial General Ledger record (fin_gl_rec)
- Fiscal Calendar record (fscl_cal_rec)
- General Ledger Account record (gla_rec)
- General Ledger Amount record (gl_amt_rec)
- General Ledger Entry record (gle_rec)
- General Ledger Transaction record (gltr_rec)
- Recur record (recur_rec)
- Standard Accounting Entries record (sae_rec)
- Subfund table (subfund_table)
- Subsidiary Account record (suba_rec)
- Subsidiary Archive record (subarc_rec)
- Subsidiary Balance record (subb_rec)
- Subsidiary Entry record (sube_rec)
- Subsidiary Total record (subt_rec)
- Subsidiary Transaction record (subtr_rec)
- Voucher record (vch_rec)

Alphabetical Organization

The tables and records appear in alphabetical order in this section.

What is an SQL Table?

In a relational SQL database, a table is an organized set of any kind of data, regardless of its purpose for validation or information maintenance. The basic unit of organization of a table is a column, a category of data. A table can have multiple columns, and columns typically contain multiple rows of data.

What is a Jenzabar CX Table?

CXmakes name distinctions in the usage of database tables. A *table* in CX contains information that remains static and is denoted with the *_table* extension. For example, the State table, named *st_table,* contains the list of the United States of America. On the CX menu, you can access most tables in *Table Maintenance* menus.

What is a Jenzabar CX Record?

CXmakes name distinctions in the usage of database tables. A *record* in CX is a table that contains information that changes on a regular basis and is denoted with the *_rec* extension. For example, the Alternate Address record, named *aa_rec*, contains any other addresses at which students can be contacted, such as a summer address. You access records in CX program screens, scroll screens, and PERFORM screens.

Common Tables

Programs in the General Ledger module use several tables that are also used in other modular areas of CX. The following common tables are used:

- 1099 table
- Configuration table
- Payment Form table
- Payment Terms table

Shared Records

Programs in the General Ledger module use several records that are shared by, created in, or most extensively used in other modular areas of CX. The following list contains these records and the modules and programs that relate to the records.

Note: If you make changes to schemas for the following records, you will need to reinstall each associated module.

- Addressing record (adr_rec)
  All modules
- Check Reconciliation record (chrecon_rec)
  Accounts Payable
  Cashier
- General Ledger Entry Name record (glename_rec)
  Cashier
- ID record (id_rec)
  All modules

Required Tables and Records

The following tables and records are required to run the features of the General Ledger module.

To define the Chart of Accounts:

- Defined Account record
- Function table

- Fund table
- Object table
- Subfund table

To produce standard reports:
- Combined Centers table
- Financial Statement table
- G/L Association table

To produce customized reports:
- Financial Format table
- Financial Statement G/L Account table
- Financial Statement Format record
- Financial Statement Report table
- Financial Statement Format table
- Financial Statement Set table

To post to the General Ledger:
- Amount Type table
- Document Type table
- Entry Type table
- Fiscal Calendar record
- Journal Type table

To post to subsidiary accounts:
- Subsidiary Association table
- Subsidiary Balance table
- Subsidiary table
- Subsidiary Totals table

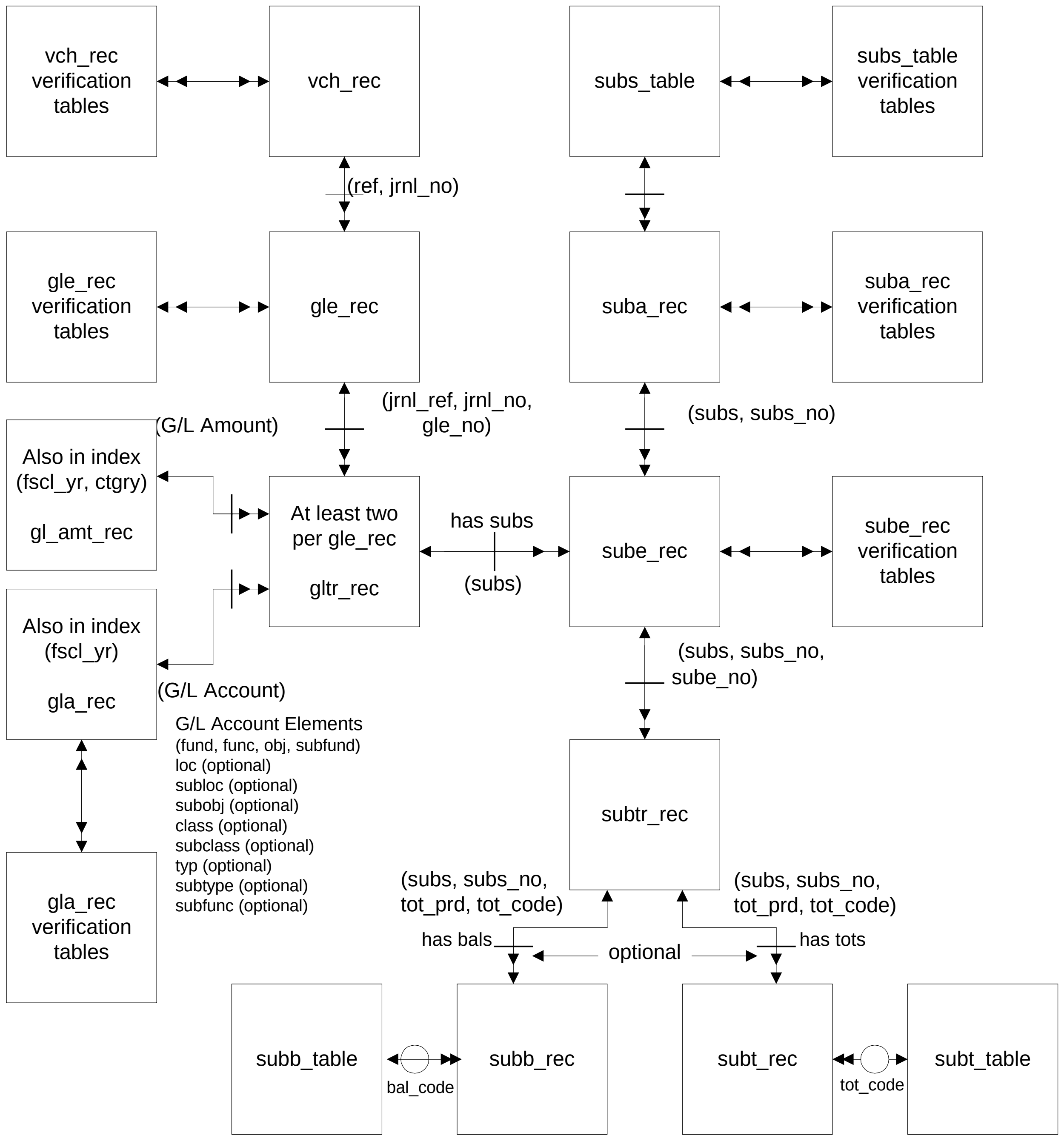
To provide security to Financial records:
- G/L Permission table
- Permission table
- User ID table

To implement the claim on cash feature:
- Claim table
- Configuration table

Table and Record Relationships

Entity Relationship Diagram

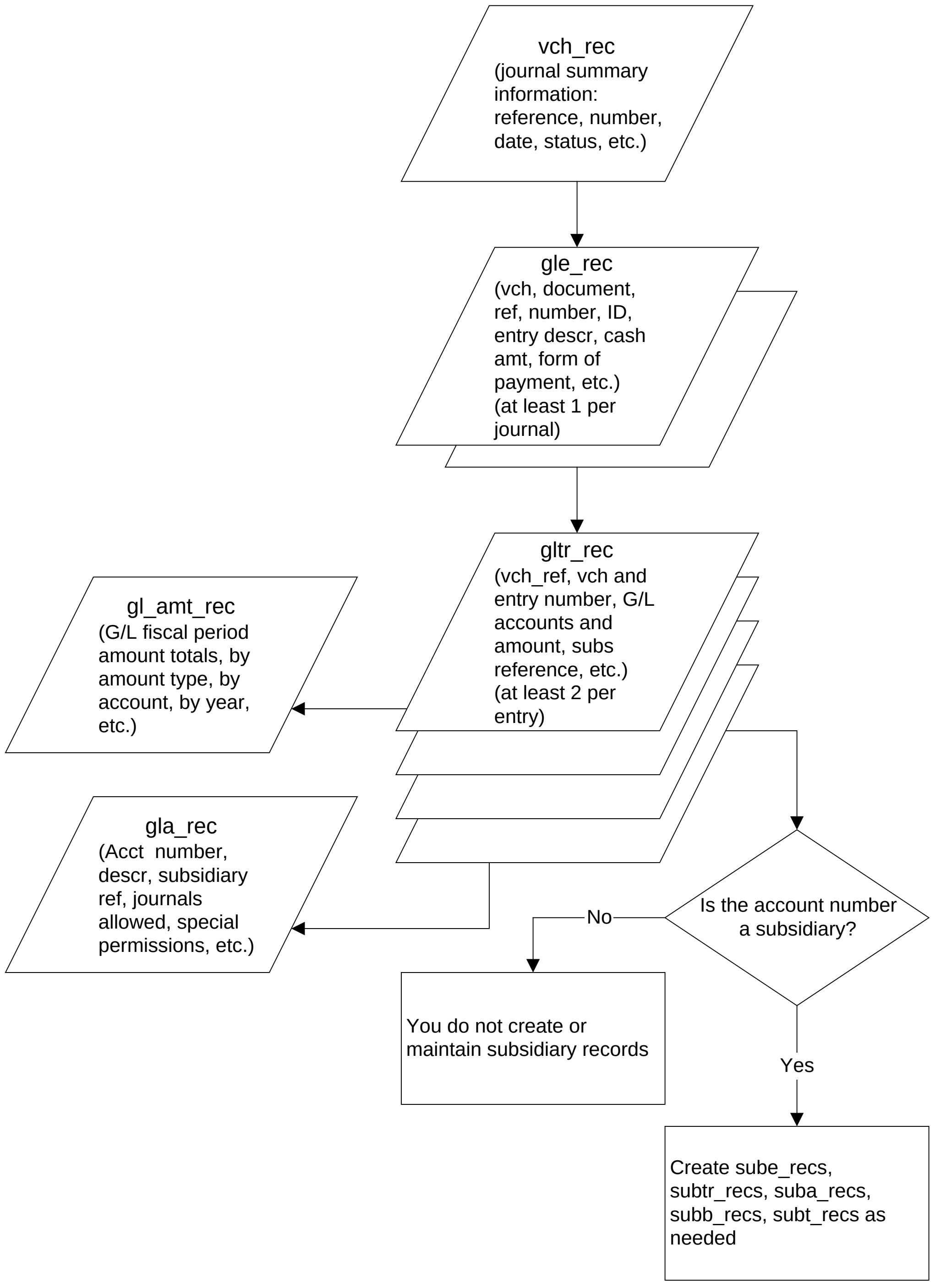
The following diagram shows the links between the tables and records used to create journal and subsidiary entries in the General Ledger module.



Record Relationships

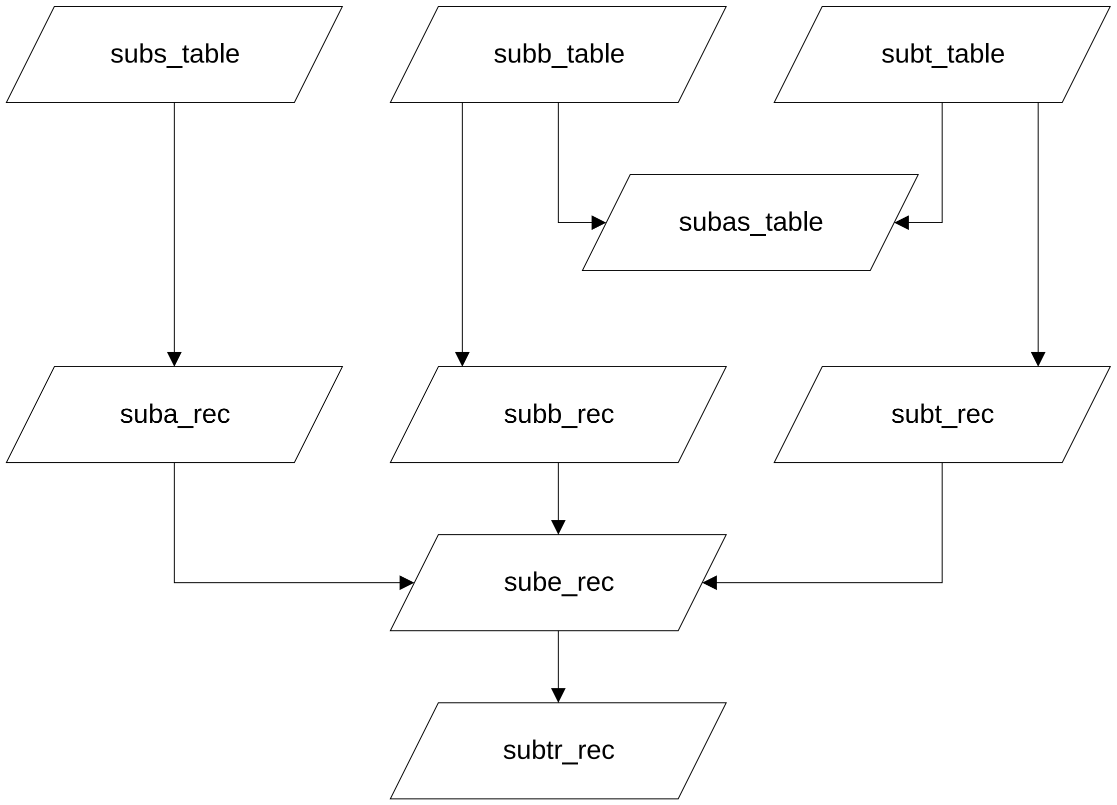
Diagram of Journal Entry Records

The following diagrams show the hierarchy and contents of the records that maintain entry information in General Ledger.


vch_rec
(journal summary information: reference, number, date, status, etc.)
gle_rec
(vch, document, ref, number, ID, entry descr, cash amt, form of payment, etc.) (at least 1 per journal)
gltr_rec
(vch_ref, vch and entry number, G/L accounts and amount, subs reference, etc.) (at least 2 per entry)
gl_amt_rec
(G/L fiscal period amount totals, by amount type, by account, by year, etc.)
gla_rec
(Acct number, descr, subsidiary ref, journals allowed, special permissions, etc.)
Is the account number a subsidiary?
No
You do not create or maintain subsidiary records
Yes
Create sube_recs, subtr_recs, suba_recs, subb_recs, subt_recs as needed

Diagrams of Subsidiary Records

The following diagrams shows the interrelationships between the records that maintain subsidiary information in General Ledger. The first diagram displays the records at a high level, and the second diagram shows the indexes and links between the records.



The following diagram displays the indexes and the common fields that link the subsidiary records.

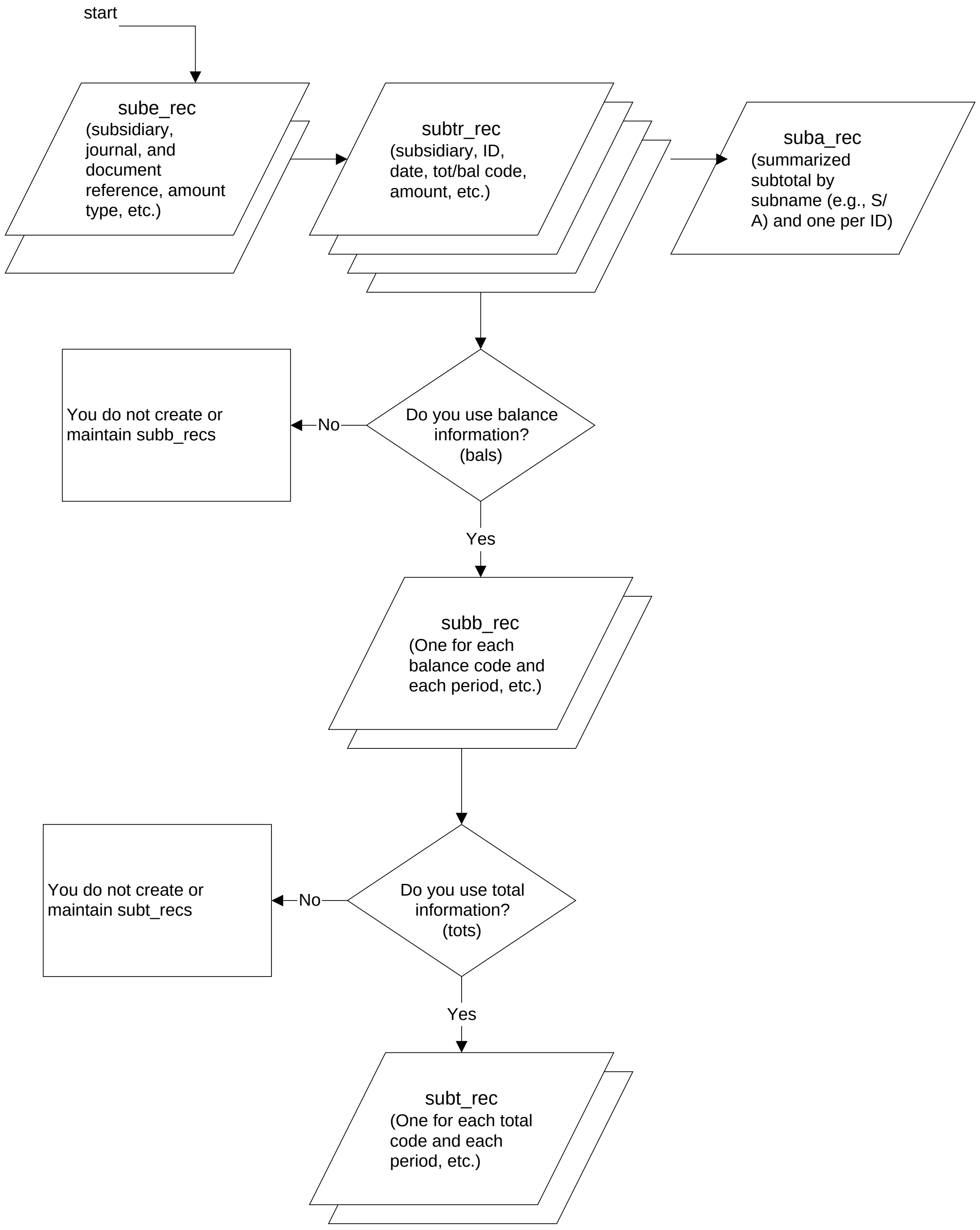

start
sube_rec
(subsidiary, journal, and document reference, amount type, etc.)
subtr_rec
(subsidiary, ID, date, tot/bal code, amount, etc.)
suba_rec
(summarized subtotal by subname (e.g., S/A) and one per ID)
Do you use balance information? (bals)
No
You do not create or maintain subb_recs
Yes
subb_rec
(One for each balance code and each period, etc.)
Do you use total information? (tots)
No
You do not create or maintain subt_recs
Yes
subt_rec
(One for each total code and each period, etc.)

Diagram of Journal Records

The following diagram shows the interrelationships between the records that the General Ledger module uses at the journal level. Note that at least one journal entry must exist in a journal.

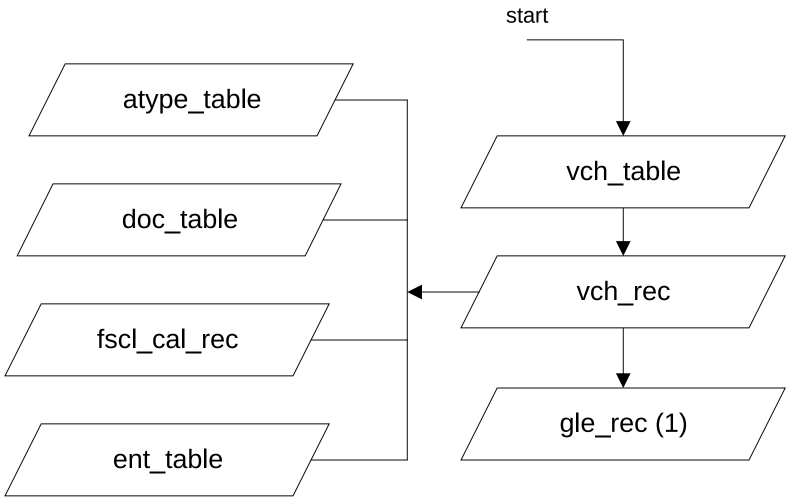
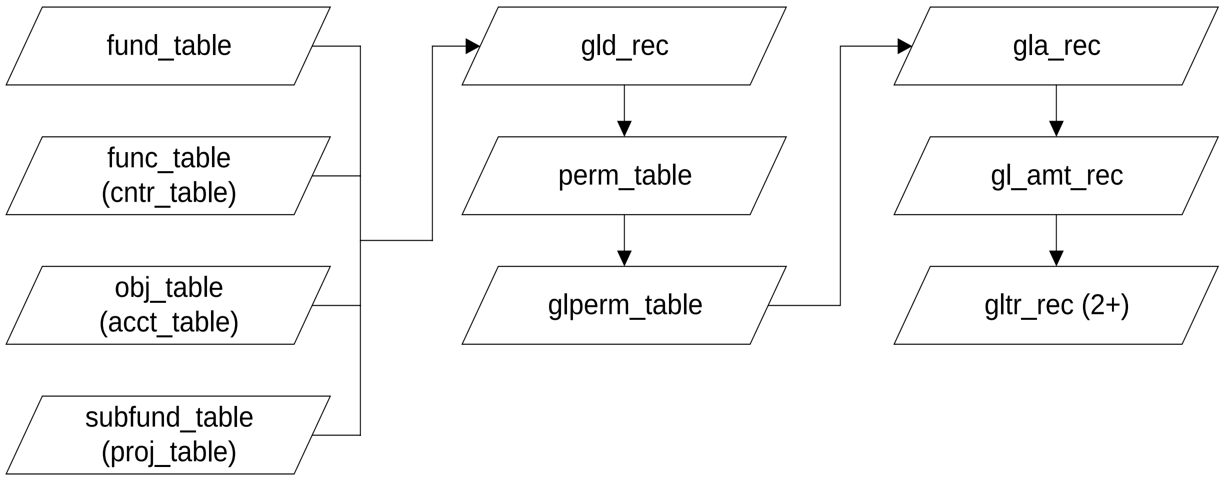


Diagram of Transaction Records

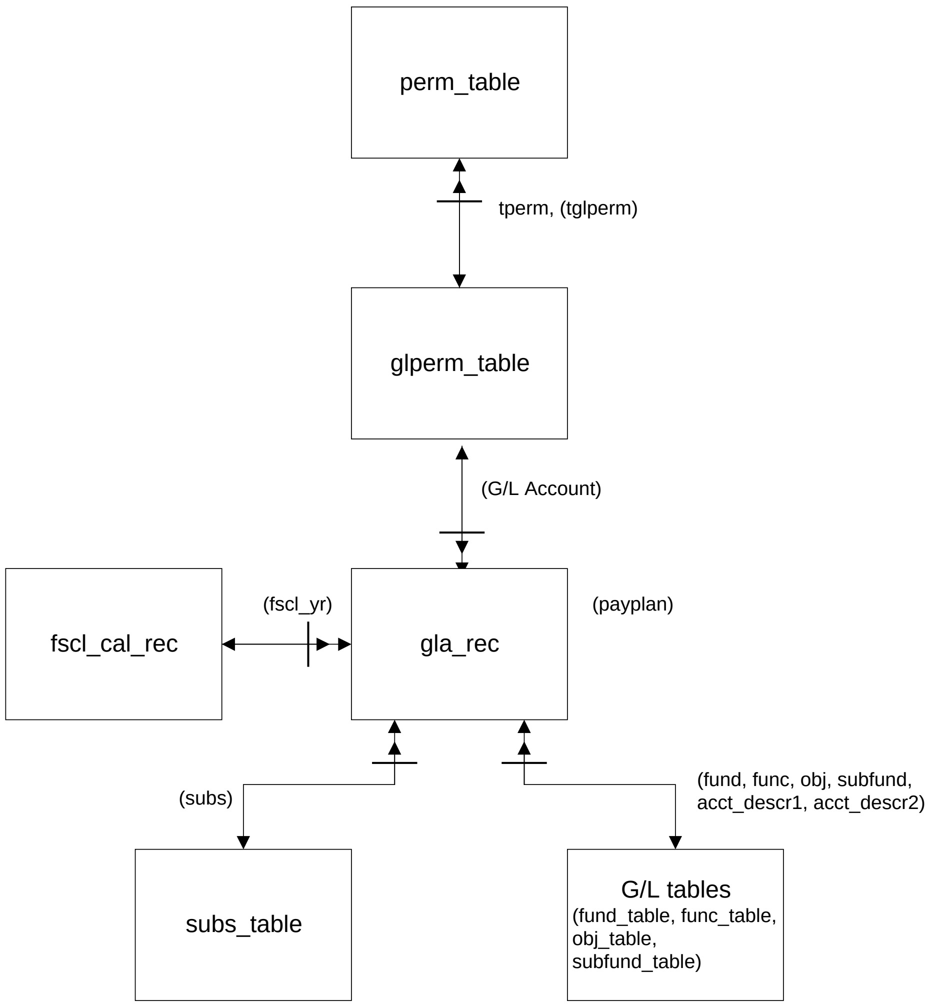
The following diagram shows the interrelationships between the records that the General Ledger module uses at the transaction level.



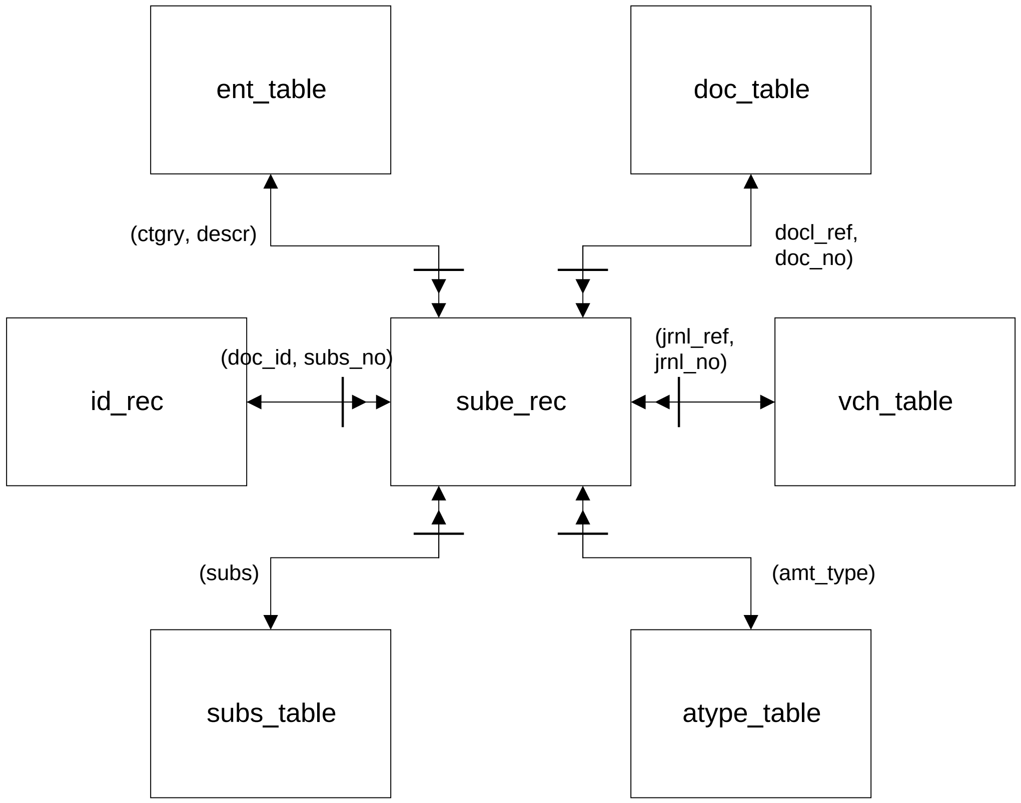
Verification Relationships

Diagrams of Verification Relationships

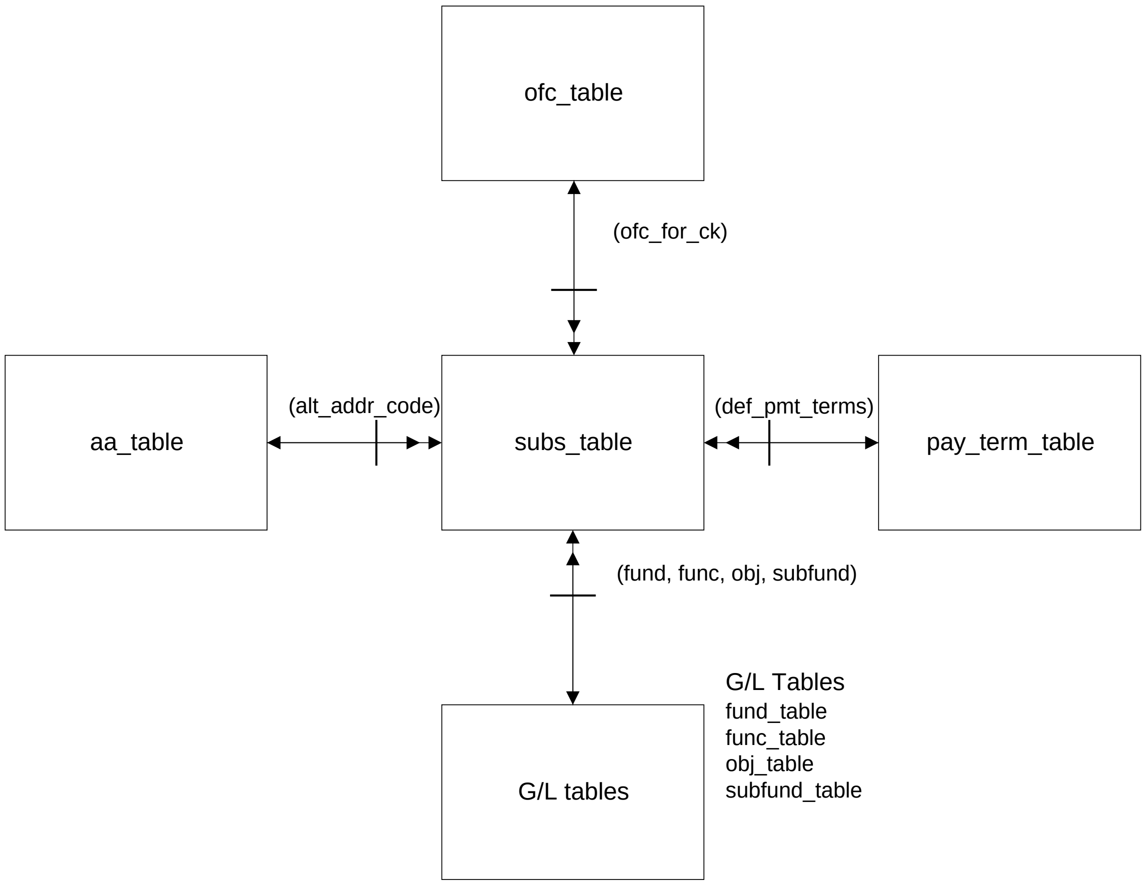
The following diagrams show the verification relationships between records in General Ledger.



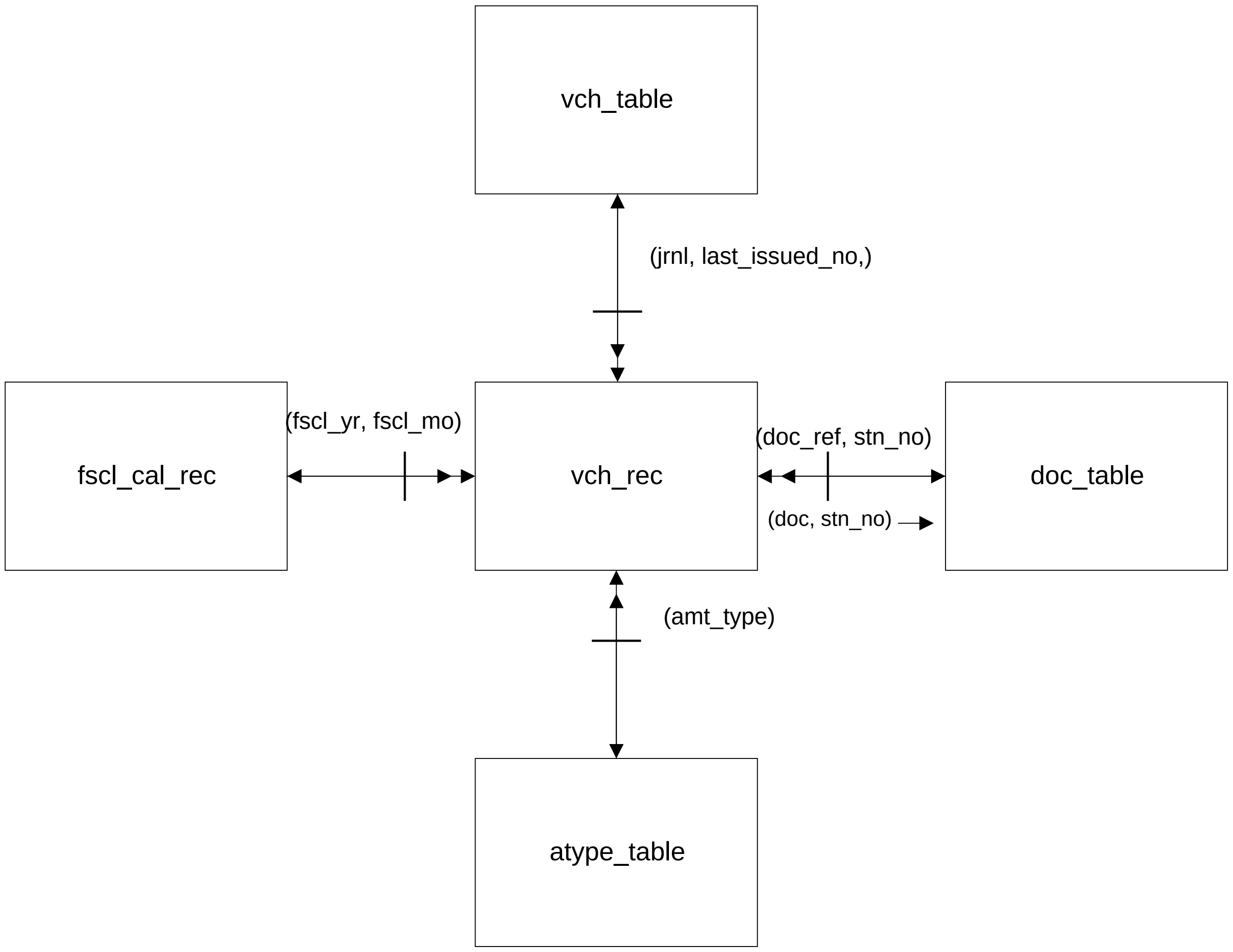
The following diagram shows the interrelationships that verify the sube_rec.



The following diagram shows the verifications performed by the subs_table.



The following diagram shows the verifications performed at the journal level.



General Ledger Schemas

Introduction

Schema files define the structure of database files and associated fields in the CX data dictionary. You can access schema files associated with the General Ledger module in the following directory path: $CARSPATH/schema/financial

File Naming Conventions

CXmakes name distinctions in the naming of schemas. For schema files containing definitions of CX tables, the UNIX filename begins with the letter *t* followed by characters describing the table's English name (e.g., tst for the State table). For schema files containing definitions of CX records, the UNIX filename describes the record's English name (e.g., as id for ID record).

The first line in a schema file, after revision information, specifies the INFORMIX database table that the schema defines. For example, *st_table* (State table) is specified in the *tst* schema file.

Field Descriptions

Schema files contain descriptions of each field defined in a table or record. You can view descriptions of fields in General Ledger tables and records by accessing the schema files.

Schema File Reports

Standard CX includes three reports that provide information about the contents of the schema files. When table implementation begins, you can run the reports to provide the installation team with information about the tables and their fields.

Select the report options from the following menu path:
System Management: Data Dictionary menu

The reports are as follows:

dbefield
Lists the columns in the database by table, including its name, short and long descriptions, field type (e.g., char or date), and size.

dbefile
Lists the tables that relate to each track area (e.g., ADM or COM), including the table name, description and purpose.

dbetrack
Combines the contents of dbefield and dbefile, displaying the tables for each track and the columns in each table.

General Ledger Tables and Records

Table and Record Information

The following alphabetical list contains the tables and records that originate from the General Ledger module. The list indicates each table/record’s purpose, its schema filename, its index and its association with programs and other tables and records.

Note: The *Program interrelationships* in the list are included in the General Ledger module.

The *Module/application interrelationships* in the list are not included in the General Ledger module.

Amount Type table
Purpose: Defines all valid amount types.
Schema filename: tatype
Informix filename: atype_table
Index type: unique simple
Index name: tatype_amt_type
Index fields: amt_type
Program interrelationships: Accounting Query, Background Voucher, General Ledger Audit, General Ledger Closing, Voucher
Module/application interrelationships: Cislib, Budget

Claim table
Purpose: Defines the accounts which can use the claim on cash feature, and the generated transactions.
Schema filename: tclaim
INFORMIX filename: claim_table
First index type: unique
First index name: tclaim_prim
First index fields: GL_FIELDS
Second index type: unique
Second index name: tclaim_contra
Second index fields: GL_FIELDS (contra_)
Program interrelationships: Background Voucher

Closing Fund Balance record
Purpose: Defines the General Ledger accounts to be closed at fiscal year-end and the fund balance accounts used for posting account balances.
Schema filename: clsgfb
INFORMIX filename: clsgfb_rec
Index type: unique
Index name: cf_prim
Index fields: The fields defined in the macro GL_FIELDS and the field fscl_yr
Program interrelationships: General Ledger Closing

Center table
Purpose: Currently not in use. Optional part of account number.
Schema filename: tcntr
Informix filename: cntr_table

Combined Centers table
Purpose: Defines center combinations for reporting purposes.
Schema filename: tcntrcomb
Informix filename: cntrcomb_table
First index type: unique

First index name: cntrcomb_prim
First index fields: the fields defined in the macro GL_FUNC_FIELD and the field cntrcomb
Second index type: simple
Second index name: cntrcomb_cntrcomb
Second index fields: cntrcomb

Defined Accounts record
Purpose: Defines rules used in validating new accounts, and contains codes used to allow single-keystroke account entry.
Schema filename: gld
INFORMIX filename: gld_rec
Index type: simple
Index name: gld_gld
Index fields: gld
Program interrelationships: Background Voucher, Voucher
Table/record interrelationships: Budget, Development, Personnel/Payroll

Document table
Purpose: Defines the types of source documents used by the system and the code that represents each document.
Schema filename: tdoc
Informix filename: doc_table
Index type: composite
Index name: tdoc_key1
Index fields: fields listed in the macro GL_FIELDS
Program interrelationships: Accounting Query, Background Voucher, General Ledger Audit, General Ledger Balance Forward, Subsidiary Account Balance Forward, Voucher Recovery, Voucher
Module/application interrelationships: Cashier, Accounts Payable, Lib, Development, Financial Aid, Payroll, Purchasing, Requisition

Entry Type table
Purpose: Defines valid entry types and the accounting functions each type may execute.
Schema filename: tent
Informix filename: ent_table
Index type: unique simple
Index name: tent_ent
Index fields: ent
Program interrelationships: Background Voucher, General Ledger Closing, Recurring Entry, Subsidiary Audit, Subsidiary Account Balance Forward, Voucher
Module/application interrelationships: Cashier, Budget, Financial Aid, Personnel/ Payroll, Purchasing, Student Billing

Financial Statement Format record
Purpose: Establishes the format for financial statements by report code, and provides columnar specifics for generating reports.
Schema filename: finrptfmt
INFORMIX filename: fin_rpt_fmt_rec
Index type: unique
Index name: finrptfmt_prim
Program interrelationships: Financial Structure Generation, Financial Report

Financial Statement G/L Account record
Purpose: Defines the General Ledger account numbers for a specific report code and fiscal year.
Schema filename: fingl
INFORMIX filename: fin_gl_rec
Index type: simple
Index name: fingl_rpt

Program interrelationships: Financial Structure Generation, Financial Report

Financial Statement Format table
Purpose: Defines valid names for report code/format code combinations.
Schema filename: tfinrptfmt
INFORMIX filename: fin_rpt_fmt_table
Index type: unique
Index name: tfinrptfmt_prim
Program interrelationships: Financial Structure Generation, Financial Report

Financial Statement Report table
Purpose: Identifies financial statements by report code. Gives report title and keeps track of fmt_no values from fin_fmt_rec.
Schema filename: tfin
Informix filename: fin_rpt_table
Index type: unique
Index name: tfin_prim
Index fields: rpt
Program interrelationships: Financial Statement Generation, Financial Report

Financial Statement Report Format table
Purpose: Contains the actual Block/Group/Schedule/Item configurations for each individual report.
Schema filename: fin
INFORMIX filename: fin_fmt_rec
Index type: simple
Index name: fin_rpt
Index fields: rpt_code
Program interrelationships: Financial Structure Generation, Financial Report

Financial Statement table
Purpose: Organizes blocks, groups, and schedules of accounts and centers on the financial statements.
Schema filename: tfs
Informix filename: fs_table
Index type: unique simple
Index name: tfs_prim
Index fields: no_name, beg_no, grp_code

Financial Statement Set table
Purpose: Identifies sets of accounts to use with customized financial statements. The sets, or groupings, provide flexibility in creating a variety of reports when used in conjunction with Financial Report Statement records.
Schema filename: tfinset
INFORMIX filename: fin_set_table
Index type: unique
Index name: tfinset_prim
Program Interrelationships: Financial Structure Generation, Financial Report

Fiscal Calendar record
Purpose: Contains valid fiscal posting periods and their valid beginning/ending and opening/closing dates.
Schema filename: fsclcal
INFORMIX filename: fscl_cal_rec
First index type: composite
First index name: fc_key3
First index fields: amt_type, fscl_mo, fscl_yr
Second index type: composite
Second index name: fc_key2

Second index fields: subs, fscl_yr, fscl_mo
Third index type: unique composite
Third index name: fc_key1
Third index fields: subs, prd, amt_type
Fourth index type: unique composite
Fourth index name: fc_prim
Fourth index fields: subs, amt_type, beg_date, prd
Fifth index type: simple
Fifth index name: fc_prd
Fifth index fields: prd
Program interrelationships: Accounting Query, Budget Review, Background Voucher, General Ledger Audit, General Ledger Balance Forward, General Ledger Closing, Recurring Entry, Standard Accounting Entries, Subsidiary Account Balance Forward, Voucher
Module/application interrelationships: Lib, Cashier, Accounts Payable, Budget, Financial Aid, Fixed Assets, Personnel/Payroll, Telephone Billing, Purchasing, Registration, Student Billing, Requisition

Function table
Purpose: Defines valid function codes.
Schema filename: tfunc
Informix filename: func_table
Index type: unique
Index name: tfunc_tfunc
Index fields: func
Program interrelationships: Voucher
Module/application interrelationships: Accounts Payable, Budget

Fund table
Purpose: Defines valid fund codes.
Schema filename: tfund
Informix filename: fund_table
Index type: unique simple
Index name: tfund_fund
Index fields: fund
Module/application interrelationships: Accounts Payable, Budget

General Ledger Account Temporary record
Purpose: Contains validation information for reporting.
Schema filename: glatemp
Informix filename: glatemp_rec
First index type: simple
First index name: glatemp_key2
First index fields: fields in the macro GL_FIELDS
Second index type: unique
Second index name: glatemp_prim
Second index fields: fields in the macro GL_FIELDS, and the fields fscl_yr, ctgry
Third index type: simple
Third index name: glatemp_fscl_yr
Third index fields: fscl_yr
Fourth index type: simple
Fourth index name: GL_FIELD_INDEXES
Fourth index fields: glatemp

General Ledger Account record
Purpose: Contains valid General Ledger accounts and validation information for the account.
Schema filename: gla
INFORMIX filename: gla_rec
First index type: unique composite

First index name: gla_prim
First index fields: the fields defined in the macro GL_FIELDS and the field fscl_yr
Second index type: composite
Second index name: gla_key1
Second index fields: the fields defined in the macro GL_FIELDS and the field fscl_yr
Third index type: simple
Third index fields: gla
Fourth index type: unique simple
Fourth index name: gla_no
Fourth index fields: gla_no
Program interrelationships: Accounting Query, Budget Review, Background Voucher, Financial Statement Generation, General Ledger Audit, General Ledger Balance Forward, General Ledger Closing, General Ledger Closing Check, Recurring Entry, Voucher
Module/application interrelationships: Lib, Cashier, Accounts Payable, Budget, Fee Collection, Financial Aid, Fixed Assets, Form Entry, Personnel/Payroll, Purchasing, Requisition

General Ledger Amount record
Purpose: Contains monthly total amounts of General Ledger accounts per account and fiscal period.
Schema filename: glamt
INFORMIX filename: glamt_rec
First index type: composite
First index name: glamt_key2
First index fields: fields defined in the macro GL_FIELDS
Second index type: composite
Second index name: glamt_key1
Second index fields: fields defined in the macro GL_FIELDS and the field fscl_yr
Third index type: unique composite
Third index name: glamt_prim
Third index fields: fields defined in the macro GL_FIELDS and the fields fscl_yr and ctgry
Fourth index type: simple
Fourth index fields: glamt
Program interrelationships: Accounting Query, Background Voucher, Budget Review, Financial Report, General Ledger Audit, General Ledger Balance Forward, General Ledger Closing, Standard Accounting Entries, Voucher
Module/application interrelationships: Lib, Accounts Payable, Budget, Purchasing, Requisition

General Ledger Association record
Purpose: Associates a code with a set of accounts for reporting purposes.
Schema filename: glas
INFORMIX filename: glas_rec
First index type: simple
First index name: glas_key2
First index fields: glas, ctgry
Second index type: simple
Second index name: glas_key1
Second index fields: GL_FIELDS
Third index type: unique
Third index name: glas_prim
Third index fields: glas, GL_FIELDS
Fourth index type: simple
Fourth index name: glas_ctgry
Fourth index fields: ctgry

General Ledger Association table
Purpose: Defines valid association codes.
Schema filename: tglas

Informix filename: glas_table
First index type: unique
First index name: tglas_prim
First index fields: glas, ctgry
Second index type: simple
Second index name: tglas_ctgry
Second index fields: ctgry

General Ledger Entries record
Purpose: Contains information relating to groups of general ledger transactions.
Schema filename: gle
Informix filename: gle_rec
First index type: composite
First index name: gle_key
First index fields: doc_ref, doc_no
Second index type: unique composite
Second index name: gle_prim
Second index fields: jrnl_ref, jrnl_no, gle_no
Program interrelationships: Accounting Query, Background Voucher, Budget Review, General Ledger Archive, General Ledger Audit, Subsidiary Audit, Voucher Recovery, Voucher
Module/application interrelationships: Cashier, Accounts Payable, Form Entry, Development, Fixed Assets, Personnel/Payroll, Purchasing, Requisition

General Ledger Permission table
Purpose: Defines general ledger account permissions, enabling users to access the applications they can use.
Schema filename: tglperm
Informix filename: glperm_table
Index type: simple
Index name: tglperm_tglperm
Index fields: tglperm

General Ledger Transactions record
Purpose: Contains individual account transactions, corresponding to individual entries in a manual accounting ledger. Must include at least one debit and one credit entry for each General Ledger Entry record (gle_rec).
Schema filename: gltr
Informix filename: gltr_rec
First index type: composite
First index name: gltr_key2
First index fields: jrnl_ref, jrnl_no, ent_no
Second index type: composite
Second index name: gltr_key1
Second index fields: the fields of the macro GL_FIELDS
Third index type: unique simple
Third index name: gltr_no
Third index fields: gltr_no
Program interrelationships: Accounting Query, Background Voucher, Budget Review, General Ledger Archive, General Ledger Audit, Subsidiary Audit, Voucher Recovery, Voucher
Module/application interrelationships: Lib, Cashier, Accounts Payable, Development, Fixed Assets, Personnel/Payroll, ADP, Requisition, Purchasing

Journal (Voucher) record
Purpose: Contains information relating to groups of general ledger entries.
Schema filename: vch
Informix filename: vch_rec
First index type: composite
First index name: vch_key2

First index fields: amt_type, fscl_mo, fscl_yr
Second index type: composite
Second index name: vch_key1
Second index fields: prep_uid, ref, stat
Third index type: unique composite
Third index name: vch_prim
Third index fields: ref, jrnl_no
Fourth index type: simple
Fourth index name: vch_fscl_yr
Fourth index fields: fscl_yr
Fifth index type: simple
Fifth index name: vch_fscl_mo
Fifth index fields: fscl_mo
Sixth index type: simple
Sixth index name: vch_jrnl_no
Sixth index fields: fscl_jrnl_no
Program interrelationships: Accounting Query, Background Voucher, Budget Review, General Ledger Archive, General Ledger Audit, General Ledger Balance Forward, General Ledger Closing, Recurring Entry, Subsidiary Audit, Voucher Recovery, Voucher
Module/application interrelationships: Lib, Cashier, Accounts Payable, Budget, Form Entry, Development, Fixed Assets, Personnel/Payroll, Purchasing, Student Billing

Journal (Voucher) table
Purpose: Defines the valid journal codes, and gives other information (permissions and restrictions) used in processing a journal.
Schema filename: tvch
Informix filename: vch_table
Index type: simple
Index name: tvch_jrnl
Index fields: jrnl
Program interrelationships: Accounting Query, Background Voucher, General Ledger Balance Forward, General Ledger Closing, Recurring Entry, Voucher
Module/application interrelationships: Accounts Payable, Personnel/Payroll

Object table
Purpose: Defines valid object codes.
Schema filename: tobj
Informix filename: obj_table
Index type: unique simple
Index name: tobj_obj
Index fields: obj
Program interrelationships: Accounts Payable, Budget

Recurring Journal Entries record
Purpose: Defines recurring journal entries.
Schema filename: recur
Informix filename: recur_rec
Index type: simple
Index name: recur_recur
Index fields: recur
Program interrelationships: Recurring Entry

Recurring Journal Entries table
Purpose: Identifies recurring journal entries and contains information required for posting.
Schema filename: trecur
Informix filename: recur_table
Index type: simple
Index name: trecur_recur

Index fields: recur
Program interrelationships: Recurring Entry

Standard Acctg Entries record
Purpose: Defines automatic periodic accounting entries.
Schema filename: sae
Informix filename: sae_rec
Index type: simple
Index name: sae_sae
Index fields: sae
Program interrelationships: Standard Accounting Entries

Std Acctg Entry table
Purpose: Defines standard accounting entry types.
Schema filename: tsae
Informix filename: sae_table
Index type: unique
Index name: tsae_sae
Index fields: sae
Program interrelationships: Standard Accounting Entries

Subfunction table
Purpose: Defines valid subfunction codes. Currently not in use. Optional part of account number.
Schema filename: tsubfunc
Informix filename: subfunc_table
Index type: unique simple
Index name: tsubfunc_subfunc
Index fields: subfunc
Program interrelationships: Accounts Payable, Budget

Subfund table
Purpose: Defines valid subfund codes.
Schema filename: tsubfund
Informix filename: subfund_table
Index type: unique simple
Index name: tsubfund_subfund
Index fields: subfund
Module/application interrelationships: Accounts Payable, Budget

Subobject table
Purpose: Defines valid sub-object codes. Currently not in use. Optional part of account number.
Schema filename: tsubobj
Informix filename: subobj_table
Index type: unique simple
Index name: tsubobj_subobj
Index fields: subobj
Module/application interrelationships: Accounts Payable, Budget

Subsidiary Account record
Purpose: Contains amount totals and other information used in subsidiary processing.
Schema filename: suba
Informix filename: suba_rec
First index type: unique
First index name: suba_prim
First index fields: subs, suba_no
Second index type: simple
Second index name: suba_id
Second index fields: id

Third index type: simple
Third index name: suba_subs
Third index fields: subs
Program interrelationships: Subsidiary Account Query, Background Voucher, Subsidiary Audit, Subsidiary Archive
Module/application interrelationships: Lib, Cashier, Accounts Payable, Form Entry, Personnel/Payroll, Purchasing, Student Billing, Requisition

Subsidiary Archive record
Purpose: Contains copies of subsidiary information or data that are to be removed from the system and archived.
Schema filename: sar
Informix filename: sar_rec
Index type: unique
Index name: sar_no
Index fields: sar_no
Program interrelationships: Subsidiary Archive
Module/application interrelationships: Financial Aid

Subsidiary Association table
Purpose: Defines valid associations between subsidiaries, subsidiary balance codes, and subsidiary total codes for reporting purposes.
Schema filename: tsubas
Informix filename: subas_table
Index type: unique
Index name: tsubas_prim
Index fields: ctgry, bal_code, tot_code

Subsidiary Balance record
Purpose: Summary information per period for subsidiary accounts or invoices for accounts payable subsidiary accounts.
Schema filename: subb
Informix filename: subb_rec
First index type: simple
First index name: subb_key1
First index fields: subs, stat, subb_no
Second index type: simple
Second index name: subb_key
Second index fields: subs, subs_no, stat
Third index type: unique
Third index name: subb_prim
Third index fields: subs, subs_no, prd, bal_code
Fourth index type: simple
Fourth index name: subb_subs_no
Fourth index fields: subs_no
Fifth index type: simple
Fifth index name: subb_subs_no
Fifth index fields: subb_no
Program interrelationships: Subsidiary Account Query, Background Voucher, Bursar, Subsidiary Audit, Subsidiary Account Balance Forward, Subsidiary Archive, Subsidiary Balance Status, Voucher
Module/application interrelationships: Lib, Accounts Payable, Personnel/Payroll, Purchasing, Student Billing, Requisition

Subsidiary Balance table
Purpose: Defines the valid subsidiary balance codes.
Schema filename: tsubb
Informix filename: subb_table

Index type: unique
Index name: tsubb_bal_code
Index fields: bal_code
Program interrelationships: Subsidiary Account Query, Background Voucher, Subsidiary Account Balance Forward, Voucher
Module/application interrelationships: Personnel/Payroll, Student Billing

Subsidiary Entry record
Purpose: Contains information about subsidiary entries posted to subsidiary accounts.
Schema filename: sube
Informix filename: sube_rec
First index type: simple
First index name: sube_key4
First index fields: doc_ref, doc_no
Second index type: simple
Second index name: sube_key1
Second index fields: jrnl_ref, jrnl_no, jrnl_ent_no
Third index type: unique
Third index name: sube_prim
Third index fields: subs, subs_no, sube_no
Program interrelationships: Subsidiary Account Query, Background Voucher, Subsidiary Audit, Subsidiary Archive, Voucher Recovery, Voucher
Module/application interrelationships: Lib, Cashier, Accounts Payable, Personnel/Payroll, Purchasing, Student Billing, Requisition

Subsidiary table
Purpose: Defines the valid subsidiary codes.
Schema filename: tsubs
Informix filename: subs_table
Index type: unique
Index name: tsubs_subs
Index fields: subs
Program interrelationships: Subsidiary Account Query, Background Voucher, Bursar, File Posting, Recurring Entry, Subsidiary Audit, Subsidiary Account Balance Forward, Subsidiary Archiving, Voucher
Module/application interrelationships: Lib, Cashier, Financial Aid, Accounts Payable, Personnel/Payroll, Purchasing, Student Billing, Requisition

Subsidiary Total record
Purpose: Contains one view of summarization information for a subsidiary. For example, in Personnel/Payroll, contains year-to-date totals.
Schema filename: subt
Informix filename: subt_rec
First index type: unique
First index name: subt_prim
First index fields: subs, subs_no, prd, tot_code
Second index type: simple
Second index name: subt_tot_code
Second index fields: tot_code
Program interrelationships: Subsidiary Account Query, Background Voucher, Subsidiary Audit, Subsidiary Archive, Voucher
Module/application interrelationships: Lib, Accounts Payable, Financial Aid, Personnel/Payroll, Student Billing

Subsidiary Total table
Purpose: Defines the valid subsidiary total codes.
Schema filename: tsubt
Informix filename: subt_table

First index type: simple
First index name: tsubt_key1
First index fields: fields defined in the macro GL_FIELDS
Second index type: unique
Second index name: tsubt_tot_code
Second index fields: tot_code
Program interrelationships: Subsidiary Account Query, Background Voucher, Subsidiary Account Balance Forward, Voucher
Module/application interrelationships: Lib, Cashier, File Posting, Financial Aid, Personnel/Payroll, Student Billing

Subsidiary Transaction record
Purpose: Contains detailed transactions for subsidiary account posting.
Schema filename: subtr
Informix filename: subtr_rec
First index type: simple
First index name: subtr_key3
First index fields: subs, subs_no, bal_prd, bal_code
Second index type: simple
Second index name: subtr_key2
Second index fields: subs, subs_no, tot_prd, tot_code
Third index type: simple
Third index name: subtr_key1
Third index fields: subs, subs_no, ent_no
Fourth index type: unique
Fourth index name: subtr_no
Fourth index fields: subtr_no
Program interrelationships: Subsidiary Account Query, Background Voucher, Subsidiary Audit, Subsidiary Archive, Voucher Recovery, Voucher

Subtype table
Purpose: Defines valid subtype codes. Currently not in use. Optional part of account number*.*
Schema filename: tsubtyp
Informix filename: subtyp_table
Index type: unique
Index name: tsubtyp_subtyp
Index fields: subtyp
Module/application interrelationships: Accounts Payable, Budget

SECTION 4 – JENZABAR CX PROGRAM FILES

Overview

Introduction

This section provides reference information about the files that relate to most CX programs. By understanding the file structure and the contents of the files, you can locate most of the information you need about any program.

Program Files Detailed

This section contains details about the following files:

def.c
The def.c file contains the declaration of external variables (including structures) that must be available to all source files in the program. These variables can also be initialized in this file. As with other C source files, the files also contain comments. The makedec command uses the def.c file to create the dec.h file.

mac.h
The mac.h file contains preprocessor include and define statements, typedef statements, and structure template definition statements. This file is included in all source files during compilation through use of the dec.h file.

Note: All other files for each CX program are standard C programming files with standard components and structure.

Definition File

Every program uses a definition (def.c) file, located in the following path: $CARSPATH/src/accounting/*program name*.

The *def.c* file for a screen-oriented program can contain the following information:

- Includes for a mac.h file
- Declaration of global variables and structures used throughout the application
- Structure declarations for structure and non-structure screen binds (i.e., program buffer to screen buffer binds)
- Ring menu definitions
- Prompt line information
- Structure definitions for program parameters
- Declarations of dynamic memory (dmms, dmls, and dmlts) in relation to functionality within libdmm (the dynamic memory management package)
- Screen pointers

The def.c file for a non-screen-oriented program can contain the following information:

- Includes for a mac.h file
- Global program variables
- Includes for schema files' def.c files
- Form pointers that provide the location for forms
- Sqlda pointers that bind the file structure to the form
- dmm, dml and dmlt definitions
- Program parameters
- Declarations of functions so the compiler can handle a call of that function

Example of a def.c File

The following is an edited excerpt from the def.c file for *fingen*. It illustrates the common components of a standard CX def.c file.

Note: The legend for the file contents appears on the next page.

```
#include "mac.h
                                                                                        1
#include <schema/financial/findef.c>
#include <schema/financial/fingldef.c>
                                                                                        2
/*-----
        Global structures and variables.
-----*/
struct fin_type                   fin_rec;
struct fingl_type                 fingl_rec;
char                    fscl_yr[5];                                                     3
char                    gl_entry[2];
char                    printer[21] ;
char                    charbuff[SCR_MAXFIELD];

/*-----
        Sqlda pointers.
-----*/
struct sqlda *finda;
struct sqlda *finglda;

/*-----                                                                                 4
        Screen and form  pointers.
-----*/
SCREEN *add_scr = NULL;
SCREEN *select_scr = NULL;
SCREEN *bgsi_form = NULL;

/*-----
Non-structure screen binds.
-----*/
struct bind_type bind_list[] =
    {                                                                                   5
    { ADD_SCR, &add_scr, "fyr", CHARTYPE, (char *)fscl_yr, SCR_CBIND },
    { SELECT_SCR, &select_scr, "L", CHARTYPE, (char *)gl_entry, SCR_CBIND },
    };
int bind_size = BIND_SIZE(bind_list);

/*-----
        Main menu structure definition.
-----*/
SCR_MENUSTART(fin_menu)
    SCR_MENUOPT2(0, "Query", QRY_CMD, SCR_GMENABLE, "Query a specific statement.", NULL),
    SCR_MENUOPT2(0, "Add", ADD_CMD, SCR_GMENABLE, "Enter a new statement.", NULL),
    SCR_MENUOPT2(0, "Remove", RMV_CMD, SCR_GMENABLE, "Remove current statement.", NULL),
    SCR_MENUOPT2(0, "Exit", EXT_CMD, SCR_GMENABLE, "Exit the program.","Exit"),
    SCR_MENUOPT2(0, charbuff, '\0', SCR_GMENABLE, NULL, NULL),
SCR_MENUEND;
int fin_size = SCR_MENUSIZE(fin_menu);
                                                                                        6
#define MENU_LEVELS     (1)

DML_START(menu_dml)
    DML_LEVEL(SCR_MENU) ,
DML_END;
DML_DEF(menu_dml);

/*-----
        Prompts
-----*/
SCR_MENUSTART(add_prompt1)
        SCR_MENUOPT2(0, NULL, SCR_DONE, SCR_GMENABLE, NULL, NULL),
        SCR_MENUOPT2(0, NULL, SCR_ABORT, SCR_GMENABLE, NULL, NULL),
        SCR_MENUOPT2(0, charbuff, '\0', SCR_GMENABLE, NULL, NULL),
SCR_MENUEND;
int addpmpt1_size = SCR_MENUSIZE(add_prompt1);

/*-----
        Program parameters.
-----*/
struct param_type param_list[] =                                                        7
    {
    { 'y', fscl_yr, PRM_CHAR, 4, PRM_REQUIRED, "fiscal year", "Fiscal Year"},
    { 'p', printer, PRM_CHAR, 20, PRM_REQUIRED, "printer device", "Printer Device" },
    };
int max_params = (sizeof(param_list) / (sizeof(param_list[0])));
```

Legend for the def.c file:

1. mac.h include
2. Schema file def.c files
3. Global program variables
4. Pointers
5. Screen binds
6. Ring menu and prompt line structure definitions
7. Program parameter structure declaration

mac.h Files

Every program uses a macro header (mac.h) file, located in the following path: $CARSPATH/src/accounting/*program name*.

The *mac.h* file for a screen-oriented program can contain the following information:

- Includes related to system header files
- Includes related to CX library and other application processes
- Includes for schema files' mac.h files
- Program constant definitions (i.e., *#define* statements)
- Structure definitions

Example of a mac.h File

The following is an edited excerpt from the mac.h file for *fingen*. It illustrates the common components of a standard CX mac.h file.

```
#include <util/cars.h>
#include <util/dmm.h>                                                         1
#include <util/dbio.h>
#include <util/fps.h>

#include <schema/financial/finmac.h>                                          2
#include <schema/financial/finglmac.h>

#define ADD_SCR                 "accounting/fingen/add"
#define SELECT_SCR              "accounting/fingen/select"
#define BGSI_FORM               "accounting/fingen/bgsi"

#define ATH_CMD         'A'
#define RMV_CMD         'R'                                                   3
#define VEW_CMD         'V'

#define ADD_PROMPT      "ADD"
#define SELECT_ACCT     "SELECT ACCT"

struct bgsi_type
    {
    struct fin_type     fin_rec;
    char                line[54];
    };                                                                        4

struct gl_type
    {
    char                gl_slctd[2];
    char                new[2];
    struct gla_type     gla_rec;
    };
```

Legend for the mac.h file:

1. includes for header files
2. includes for schema files
3. program constant definitions
4. structure definitions

SECTION 5 - ACCOUNTING QUERY AND SUBSIDIARY QUERY

Overview

Introduction

This section provides reference information about the Accounting Query program (*acquery*) and the Subsidiary Query program (*saquery*). The General Ledger module uses the query programs to enable users to view account, document, journal, and subsidiary information in the database.

Program Features Detailed

This section contains details about the following features of the Accounting Query and Subsidiary Query programs:

- Process flows
- Parameters
- Table usage
- Program screens

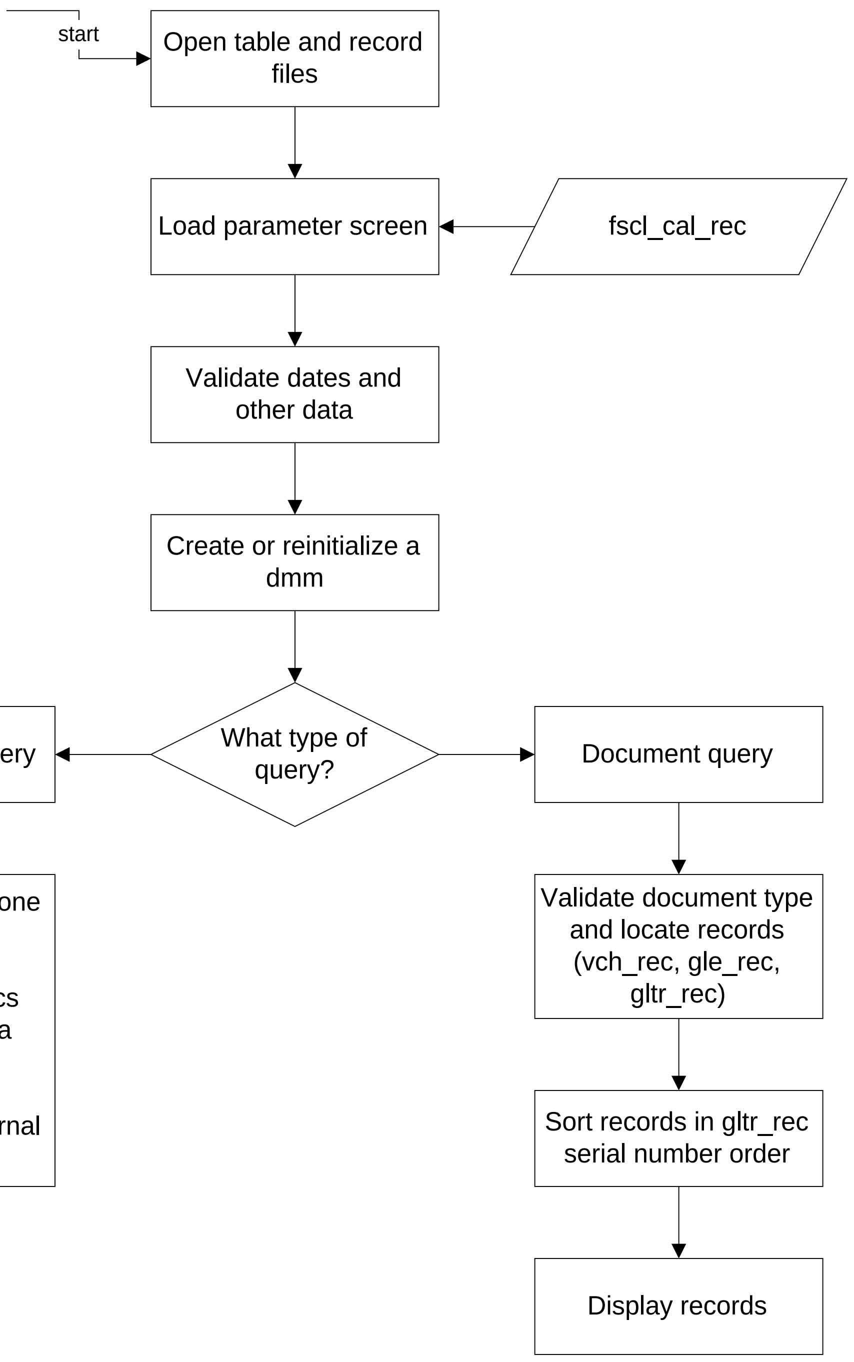
Program Files

All the program files for *acquery* are located in the following directory:
$CARSPATH/src/accounting/acquery

Process Flow

Diagram

The following diagram shows the flow of data in the *acquery* program.


start
Open table and record files
Load parameter screen
fscl_cal_rec
Validate dates and other data
Create or reinitialize a dmm
What type of query?
Journal or entry query
Document query
Display records for one of the following queries:
1. Range of vch_recs
2. All entries under a specific journal
3. Range of entries under a specific journal
4. Specific entry
Validate document type and locate records (vch_rec, gle_rec, gltr_rec)
Sort records in gltr_rec serial number order
Display records

Data Flow Description for ***acquery***

The following process describes the data flow in *acquery*.

1. The user accesses the program.

   Note: No parameters are required to run *acquery*.

2. The program opens all the required files immediately. Required files are:
   - atype_table
   - doc_table
   - fscl_cal_rec
   - gla_rec
   - glamt_rec
   - gle_rec
   - gltr_rec
   - id_rec
   - vch_rec
   - vch_table
3. The user enters query search criteria.
4. The program performs the following validations:
   - The beginning period must be valid in the fscl_cal_rec. The key to use is fc_key2, which includes subs, amt_type and month.
   - The ending period must be valid in the fscl_cal_rec. The key to use is fc_key2, which includes subs, amt_type and month.
   - The beginning period number must be less than or equal to the ending period number.
   - The status code must be one of the following:
     - A (all)
     - D (detail and posted)
     - O (offsetting)
     - P (posted)
     - R (removed)
     - S (summarized and posted)
     - V (voided)
     - U (unsummarized)
5. The program creates or reinitializes a dmm that includes all the fiscal periods for the fiscal year. The structure includes the fc_prd_no and fc_mo. Therefore the system locates all Fiscal Calendar records for periods 0-14, using the key fc_key1. The components of the key are fc_subs=G/L, fc_amt_type=parameter, and fc_prd=0. If the program cannot locate any of these Fiscal Calendar records, a fatal error occurs.
6. Based on the information entered, the system performs either a document, entry or journal query. Since the program can only take one path, the program interprets fields left to right. When the program reads information on the left part of the query line, it clears the remaining fields and takes the appropriate course of action. Conversely, when the user advances the cursor past the fields on the left part of the query line to enter information on the right part of the line, the program takes a different course of action.
7. When the user initiates a document query, the system performs a check to validate that the document type and amount type create a valid combination, according to the Document table. The system also assumes that, if the ending document number is blank, the beginning number becomes the ending number. If the ending document number is not blank, the system performs a check that the beginning number is less than the ending number.
8. For a document query, the program binds the following fields on the d1 screen:

- docno=gle_rec.doc_no
- prd=vch_rec.no
- payno=gle_rec.pay_no
- cash=gle_rec.cash_amt
- ent=gle_rec.type
- f=gle_rec.pay_form
- docid=gle_rec.doc_id
- a=gle_rec.status

The program also does a dbselfield on gle_key (gle_doc_ref, gle_doc_no) to find all general ledger entries. The system uses the GTEQ and NEXT flags to check if the document number that it locates is greater than the ending document number. The system allows multiple general ledger entries within the same document number.

9. As the program locates entries, the program also locates the corresponding vch_rec to determine if the entry relates to a period that the user entered in the parameter line. In addition, the Status from the parameter line must be validated for the entry.
10. All entries go into the dmm that is used for the display, which sorts by document number and gle date. If the program cannot locate any entries that meet the search criteria, the docgle screen does not appear, and the program continues to display the query screen.
11. When the program displays the docgle screen, the user can select from the normal screen package commands FASTF, FASTB, UP, and DOWN to move through the display. The user can also select a letter on the left column of the screen to view detail information about the corresponding document.
12. The program provides a line at the bottom of the screen that the user can use to search the data set. A generalized function allows the user to fill in any combination of fields, and then the program searches the records in the dmm for an exact match. If the program cannot locate an exact match, the search aborts. The only exception to the exact match requirement is if the document number is the only field entered; for selection purposes, the program locates numbers that are greater than or equal to the entered number.
13. The program uses the gltr screen, from any path in *acquery*. It accepts three parameters: the voucher reference, the voucher number and the entry number.
14. On the gltr screen, the program operates under the assumption that no records have been located that meet the search criteria. Because of this assumption, this function uses its own vch, gle and gltr records declared in the def.c file. These records appear in the def.c file to allow the binding of the screens to be in a separate file.
15. The program then locates the vch_rec, the gle_rec, and the gltr_recs, sorted in gltr_serial order.

    Note: Do not place the #define DISPLAY_SERIAL in the mac.h because it would require a mastermake for the purpose of finding a transaction number.
16. The user can move through the screen using the normal scroll package.
17. In the case of a journal or entry query, the program supports four types of queries:
    - Case 1: Showing a range of voucher records (on the v1 screen). See steps 18 and 19.
    - Case 2: Showing all entries under a specific journal (on the v2 screen). See step 20.
    - Case 3: Showing a range of entries under a specific journal (on the v2 screen). See step 21.
    - Case 4: Showing a specific entry (on the v3 screen). See step 22.
18. When the user specifies a query on voucher records, the program performs the following checks:
    - The vf and the vchno fields must not be blank.

- If the vchent field is not blank and the ent2 field is blank, then the status field must contain either E (for entry) or V (for voucher). If the status field contains a V, then the program checks to ensure that the beginning number is not greater than the ending number.
- If the vf and the vchno fields are the only two search criteria fields that contain data, the program assumes a status of V. If the vf, vchno, vchent and ent2 fields contain data, the program assumes a status of E.

19. If the query is Case 1, the program uses the following logic, based on the example [CK][1000][1005][ ][V]:
    - Call findvch("CK",1000,1005). This function fills the vch_dmm, and verifies the voucher record before adding more records to the dmm.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - Display v1 screen.
    - If v2 screen is required, then call findgle.
20. If the query is Case 2, the program uses the following logic, based on the example [CK][1000][ ][ ][ ]:
    - Call findvch("CK",1000,1000). This function fills the vch_dmm and verifies the voucher record before adding more records to the dmm.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - Call fingle("CK",1000,0,0). The third and fourth arguments are entry numbers; 0 signifies load all.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - If v2 screen is required, then call findgle.
21. If the query is Case 3, the program uses the following logic, based on the example [CK][1000][5][10][ ]:
    - Call findvch("CK",1000,1000). This function fills the vch_dmm and verifies the voucher record before adding more records to the dmm.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - Call fingle("CK",1000,5,10). The third and fourth arguments are entry numbers.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - If v2 screen is required, then call findgle.
22. If the query is Case 4, the program uses the following logic, based on the example [CK][1000][5][ ][E]:
    - Call findvch("CK",1000,1000). This function fills the vch_dmm and verifies the voucher record before adding more records to the dmm.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - Call fingle("CK",1000,5,5). The third and fourth arguments are entry numbers.
    - If (dmm_size(&vch_dmm) ==0). This function returns the message that no records that match the search criteria were located.
    - Call gentrs("CK",1000,5)
23. In the case of an entry query, the program performs the following steps:
    - Selects the key for the glamt_rec. If the user enters information into any of the fields in the account section of the query, then the program must evaluate each of the fields to determine which key to use on the glamt_rec, based on the following tests:
        - glamt_prim [10] [ ] [1010] [ ]

If no asterisks appear in any of the account component fields. The fiscal year and amount type appear in the parameter screen.

– glamt_key1   [10] [1220] [6***] [   ]

If the fund and function are not blank, and no asterisks appear in either the fund or function field. If the first character of the object is masked or blank, the program warns the user that the search will be time consuming, and prompts the user about continuing the search. The fiscal year appears in the parameter screen.

– glamt_prim   [10] [1***] [6110] [   ]

If the fund and object are not blank, and no asterisks appear in the fund or object field. If the first character of the function is masked or blank, the program warns the user that the search will be time consuming, and prompts the user about continuing the search. The fiscal year and the amount type appear in the parameter screen.

– glamt_key1   [10] [1***] [****] [   ]

If the fund and function are not blank, and the fund is not masked, and the first character of the function is not masked. The program warns the user that the search will be time consuming, and prompts the user about continuing the search. The fiscal year appears in the parameter screen.

– glamt_prim   [10] [****] [1***] [   ]

If the fund and object are not blank, and the fund is not masked, and the first character of the object is not masked. The program warns the user that the search will be time consuming, and prompts the user about continuing the search. The fiscal year and the amount type appear in the parameter screen.

– glamt_subfund   [10] [****] [****] [1234]

If the subfund is not blank and an asterisk does not appear in the first position of the field.

– glamt_func   [10] [1***] [****] [****]

If the function is not blank and an asterisk does not appear in the first position of the field.

– glamt_obj   [10] [****] [1****] [****]

If the object is not blank and an asterisk does not appear in the first position of the field.

– glamt_key1   [10] [****] [****] [****]

If the fund is not blank and an asterisk does not appear in the first position of the field.

– glamt_key1   [**] [**20] [****] [****]

The program warns the user that the program will perform a sequential search through the glamt_recs for the fiscal year.

1. The program locates all the glamt_recs that match the query criteria.
2. If the dmm_size (&glamt_dmm) = 0, the program informs the user that it did not locate any records that match the search criteria.
3. If the dmm_size (&glamt_dmm) > 0, the program loads all the gla_recs for the glamt_recs found for the purpose of displaying the description.
4. If the dmm_size (&glamt_dmm) = 1, the program displays the Amount screen. When the user selects **Detail of Amount**, the Transaction/Detail screen appears.
5. If the dmm_size (&glamt_dmm) > 1, the program displays the Account screen. The SCR_COMMAND followed by scr_getc of a selected letter (a-l) takes the user to the Amount screen.

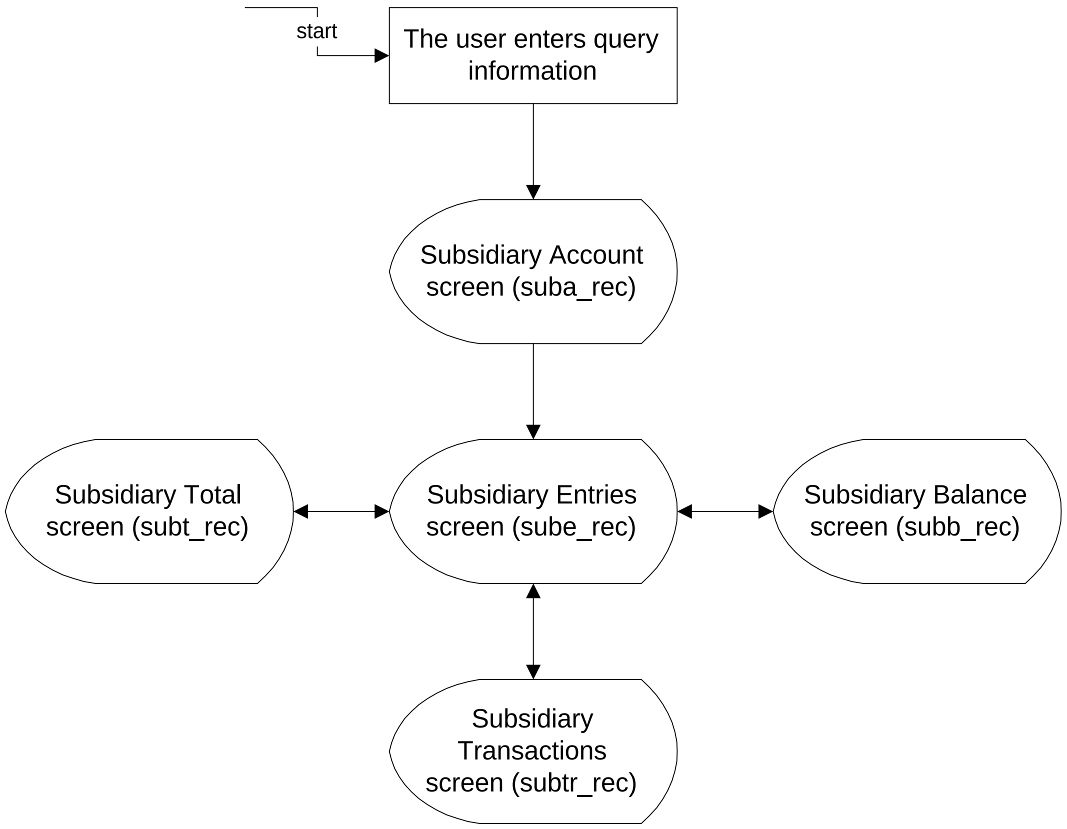
Program Interrelationships

Acquery interacts only with *saquery*.

Process Flow for Subsidiary Query

Diagram

The following diagram shows the flow of data in the *saquery* program.


start
The user enters query information
Subsidiary Account screen (suba_rec)
Subsidiary Total screen (subt_rec)
Subsidiary Entries screen (sube_rec)
Subsidiary Balance screen (subb_rec)
Subsidiary Transactions screen (subtr_rec)

Data Flow Description for ***saquery***

The *saquery* program enables users to view subsidiary information. The method of access for *saquery* is through the command line in *acquery*, or by entering the -s argument on the command line.

The program loads the following tables as needed. The tables marked with an asterisk (*) are loaded into memory; that is, the program does not look them up dynamically from the disk.

- atype_table*
- doc_table*
- fscl_cal_rec*
- gla_rec
- glamt_rec
- gle_rec
- gltr_rec
- id_rec
- suba_rec
- subas_table*
- subb_rec
- subb_table*
- sube_rec
- subs_table*
- subt_rec
- subt_table*
- subtr_rec
- vch_rec
- vch_table*

The following list describes the program flow in *saquery*.

1. The user accesses *saquery*.
2. If only one record satisfies the search criteria, the program displays the Subsidiary Entries screen. If more than one record satisfies the search criteria, the program displays the Subsidiary Accounts screen.
3. If the Subsidiary Accounts screen appears, the user can view bal or tot information by changing the Level code. This information appears on the Subsidiary Balance screen or the Subsidiary Total screen.
4. The most detailed level of query appears on the Subsidiary Transaction screen.
5. Types of queries, the result, and the keys used are as follows:
   - Subsidiary only
     - If the program locates more than one suba_rec, the Subsidiary Account screen appears. If the program locates only one suba_rec, the Subsidiary Entries screen appears.
     - The key used is suba_subs.
   - Subsidiary number only
     - If the program locates more than one suba_rec, the Subsidiary Account screen appears. If the program locates only one suba_rec, the Subsidiary Entries screen appears.
     - The key used is suba_id.
   - Subsidiary and subsidiary number
     - Displays the level specified on the parameter screen.

    - The key used is suba_prim, followed by subb_key (subb_subs=suba_subs, subb_no=suba_no, subt_key (subt_subs=suba_subs, subt_no=suba_no), sube_prim (sube_subs=suba_subs, sube_subs_no=suba_no), or nothing
- Subsidiary, subsidiary number and bal code
    - Displays the Subsidiary Balance screen, but can also go to the Subsidiary Account screen, the Subsidiary Totals screen, or the Subsidiary Entries screen if specified)
    - The key used is suba_prim, followed by subb_key, subt_key, or sube_prim.
- Subsidiary, subsidiary number and tot code
    - Displays the Subsidiary Totals screen, but can also go to the Subsidiary Account screen, the Subsidiary Balance screen, or the Subsidiary Entries screen if specified.
    - The key used is suba_prim, followed by subb_key, subt_key, or sube_prim.

Program Interrelationships

Saquery interacts only with *acquery*.

Query Program Parameters

Introduction

CX contains parameters and compilation values for executing the query programs. You can specify parameters to use *acquery* or *saquery* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the query programs.

Parameter Syntax

You can display *acquery* parameters by entering the following: **acquery -,**

Because you access *saquery* from within *acquery*, no parameters relate to *saquery*.

The following is the correct usage for running the Accounting Query program from the UNIX shell:

```
acquery [-l]  [-s [sub]]  [-r sub1 [sub2 ...]]  [-p printer]
```

Parameters that appear in brackets are optional. For *acquery*, all parameters are optional.

Parameters

The following lists the parameters for running *acquery*.

-l
Optional - Restricts users to using either Accounting Query or Subsidiary Query, prohibiting them from crossing between programs.

-p printer
Optional - Specifies a default printer.

-r sub1 (and other subsidiaries as required)
Optional - Specifies the name of a subsidiary. If none is entered, the program does not restrict the user to a specific subsidiary. You can enter multiple subsidiaries.

-s sub
Optional - Initiates the *saquery* program from the UNIX command line, using the specified subsidiary.

Program Screens

Introduction

The standard CX *acquery* and *saquery* programs are delivered with 31 screen files, including some files that contain variations (or toggle options) from the original screens.

Access

The screen files are located in the following directory path:
$CARSPATH/src/accounting/acquery/SCR

Screen Files

The query screens appear in the following files:

acctgle
Contains the Entry Detail screen
Table/Records: gla_rec, gltr_rec, gle_rec

docgle
Contains the Document Query screen
Table/Records: gle_rec

fscl
Contains the Fiscal Calendar Lookup window
Table/Records: fscl_cal_rec

glacct
Contains the Account screen
Table/Records: gla_rec

glamt
Contains the Amount screen
Table/Records: gla_rec

glparam
Contains the parameter line portion of the Accounting Query Parameter screen
Table/Records: fscl_cal_rec, atype_table

glparmhelp
Contains help information for the parameter line of the Accounting Query Parameter screen

glquerhelp
Contains help information for the query line of the Accounting Query Parameter screen

glquery
Contains the query line portion of Accounting Query Parameter screen
Table/Records: fscl_cal_rec, atype_table

gltr
Contains the Transaction/Detail screen
Table/Records: vch_rec, gle_rec, gla_rec, gltr_rec

prtchoice
Contains a printer selection screen

saparam

Contains the parameter line of the Subsidiary Query Parameter screen
Table/Records: subs_table, atype_table

saparmhelp
Contains help information for the parameter line of the Subsidiary Query Parameter screen

saquerhelp
Contains help information for the query line of the Subsidiary Query Parameter screen

saquery
Contains the query line of the Subsidiary Query Parameter screen
Table/Records: subs_table, subt_table, subb_table, vch_table, doc_table

stmt
Contains the Statement screen
Table/Records: subs_table, subtr_rec, gltr_rec

suba
Contains the Subsidiary Accounts screen
Table/Records: suba_rec

subb
Contains the Subsidiary Balances screen
Table/Records: suba_rec, subb_rec

subbtoggle
Contains a variation of the Subsidiary Balances screen
Table/Records: suba_rec, subb_rec

sube1
Contains the Subsidiary Entries screen
Table/Records: suba_rec, sube_rec

sube2
Contains a variation of the Subsidiary Entries screen
Table/Records: suba_rec, sube_rec

subedoc1
Contains a variation of the Subsidiary Entries screen
Table/Records: sube_rec

subedoc2
Contains a variation of the Subsidiary Entries screen
Table/Records: sube_rec

subevch1
Contains a variation of the Subsidiary Entries screen
Table/Records: sube_rec

subevch2
Contains a variation of the Subsidiary Entries screen
Table/Records: sube_rec

subt
Contains the Subsidiary Totals screen
Table/Records: suba_rec, subt_rec

subtr
Contains the Subsidiary Transactions screen
Table/Records: subtr_rec, sube_rec

subtrser

Contains a variation of the Subsidiary Transactions screen
Table/Records: subtr_rec, sube_rec

truesuba
Contains the Subsidiary Account screen
Table/Records: suba_rec

vch
Contains the Journal screen
Table/Records: vch_rec

vchgle
Contains the Entry List screen
Table/Records: vch_rec, gle_rec

SECTION 6 - BACKGROUND VOUCHER

Overview

Introduction

This section provides reference information about the Background Voucher (*bgvoucher*) program. The General Ledger module uses *bgvoucher* to perform all the file handling for programs that create accounting input.

Program Features Detailed

This section contains details about the following features of the *bgvoucher* program:

- Process flow
- Table usage
- Parameters

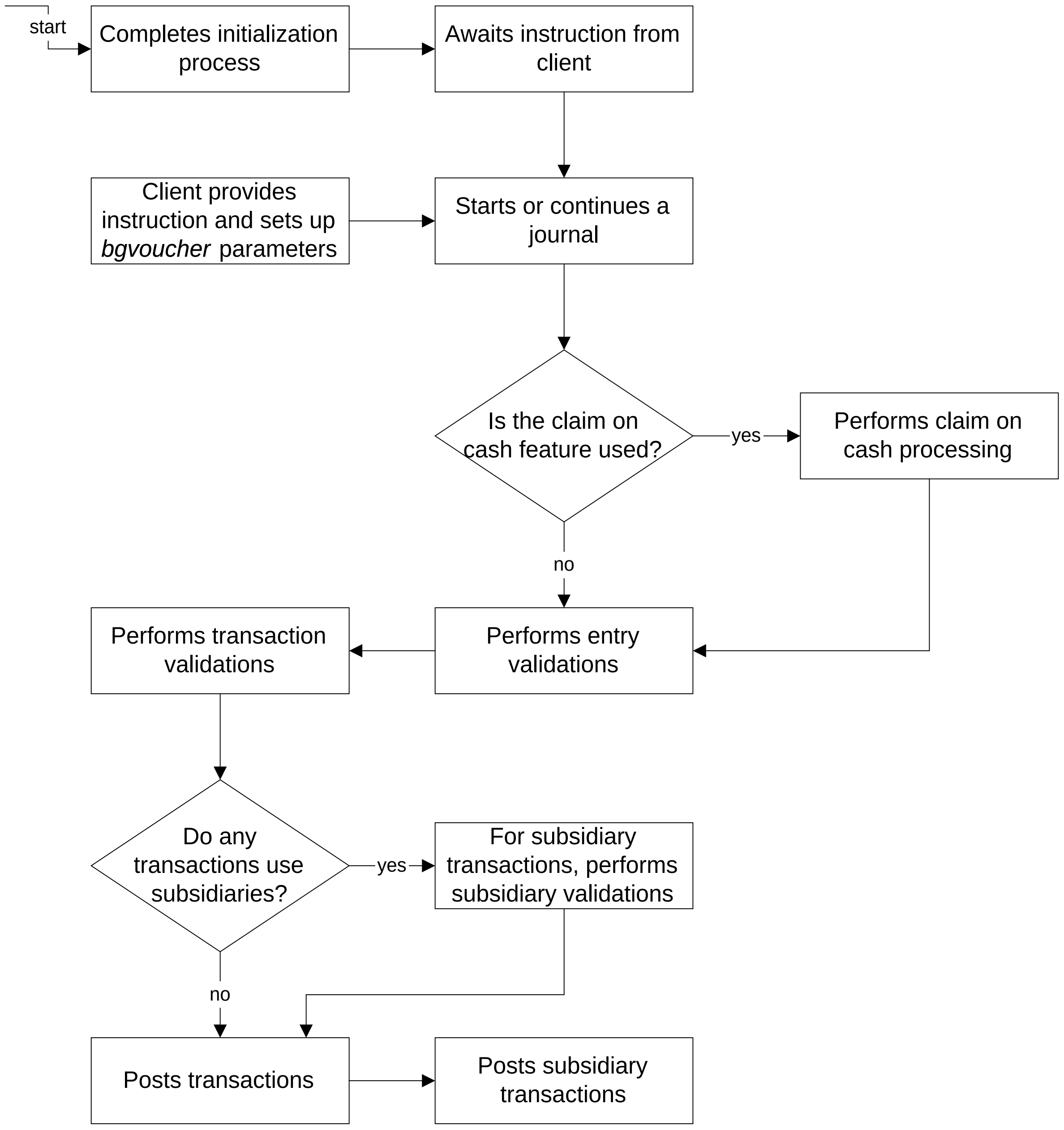
Program Files

All the program files for *bgvoucher* appear in the following directory:
$CARSPATH/src/accounting/bgvoucher

Process Flow

Diagram

The following diagram shows the flow of data in the *bgvoucher* program.


start
Completes initialization process
Awaits instruction from client
Client provides instruction and sets up *bgvoucher* parameters
Starts or continues a journal
Is the claim on cash feature used?
yes
Performs claim on cash processing
no
Performs transaction validations
Performs entry validations
Do any transactions use subsidiaries?
yes
For subsidiary transactions, performs subsidiary validations
no
Posts transactions
Posts subsidiary transactions

Data Flow Description

The following process describes the data flow in the *bgvoucher* program.

1. The program completes the initialization process by completing the following steps:
    - Using the bgv_accept call to accept the connection from the client.
    - Adding this connection to the connection queue, setting the connection status to BGV_LOADING.
    - Initializing the interrupt routine.
    - Loading the following records into memory:
        - atype_table
        - doc_table
        - entry_table
        - fscl_cal_rec
        - fund_table
        - gld_rec
        - pay_frm_table
        - pay_term_table
        - subas_table
        - subb_table
        - subfund_table
        - subs_table
        - subt_table
    - Loading the gld_rec for due to/from accounts into a special record for use during generation of due/to from transactions, and verifies that all ent_inv_or_pay codes are INV or PAY.
    - Opening the following files:
        - gla_rec
        - glamt_rec
        - gle_rec
        - gltr_rec
        - id_rec
        - suba_rec
        - subb_rec
        - sube_rec
        - subt_rec
        - subtr_rec
        - vch_rec
    - Opening and closing the following files as needed:
        - func_table
        - obj_table
        - vch_table
    - Obtaining the system time and date.
2. The program awaits instruction from the client.

    Note: The first instruction should be getstatus, but *bgvoucher* does not require this command as the first instruction.
3. In response to the getstatus instruction, the program uses bgv_putstatus to send a BGV_LOADED status to the client program.

    Note: If the client is opening multiple connections, the first command should still be bgv_getstatus before the program tries to open a second connection. After the bgv_connect call, the client should then do another bgv_getstatus to obtain the status of the second connection.

4. The client program sets up *bgvoucher* parameters with the following function calls:
    - bgv_setnotify
    - bgv_setwrite
    - bgv_savecopy

Process Description for Starting a Journal

When the client is ready to start a journal, it uses the bgv_start function to send the journal header record to *bgvoucher*. *Bgvoucher* performs the following steps when it receives a Start command:

1. Validates the period/date combination.
2. Validates group against tvch_grp_id (i.e., if vch_table != 0, bg_head_rec.group must equal tvch_grp_id).
3. Validates the amount type for this journal in the Voucher table.
4. Updates tvch_last_issued.
5. Adds a vch_rec with an S status.

Process Description for Continuing a Journal

When the client is ready to continue an incomplete journal, it uses the bgv_continue function to send *bgvoucher* a vch_ref and vch_no. *Bgvoucher* performs the following steps when it receives a Continue command:

1. Ensures the vch_rec specified exists.
2. Ensures the vch_stat is I.
3. Ensures the current userid is the same as the vch_prep_uid (only the individual who started a specific journal can continue or complete it).
4. Ensures the period specified in the vch_rec is still open based on the system date and the Fiscal Calendar record.
5. Updates the vch_rec status to C.

Process Description for Creating Claim on Cash Entries

If the institution is using Claim on Cash, Receivables and Payables (the claim on cash feature), it uses the gen_claim_on_cash function. Bgvoucher performs the following steps when processing claim on cash:

1. Reviews each transaction and determines if any transactions affect a claim on cash account, as defined in the claim_table, and flags such transactions for processing. The check against the claim_table also includes a verification that the claim_table entry is valid for the current date (e.g., the current date falls between the Active Date and the Inactive Date, if such dates are specified).
2. Generates claim on cash entries.
3. If multiple accounts are affected (e.g., if the input transaction contains multiple expense accounts with a single offset to Accounts Payable), creates a corresponding subsidiary transaction for each input account.

Process Description for Verification

When the interrupt handler receives an instruction to read an entry dml, it adds an instruction of *verify* to the interrupt queue and reads the entries and transactions into the appropriate level in the interrupt queue. *Bgvoucher* eventually transfers this information to the connection queue, converting it to the standard gle_rec, gltr_rec, sube_rec and subtr_recs in the process.

Bgvoucher performs the following steps during the verification pass:

1. Ensures entry_table.ent_inv_or_pay is INV or PAY at entry_table load time.
2. Validates entry type in the entry_table.
3. Validates journal type (PR) against the entry type (e.g., tent_ac_allow).
4. Validates doc_id from the id file (if the id is not 0).
5. Validates document and stn_no in doc_table.
6. Validates vch_type against this document, (e.g., tdoc_ac_allow).
7. Ensures the number of transactions is 0 or greater than or equal to 2.

For each transaction in the entry, *bgvoucher* performs the following steps during the verification pass:

1. Rounds the transaction amount.
2. Validates the account in the gla_rec. If *bgvoucher* cannot locate the record, it reviews the previous year's gla_recs. If it locates the previous year's record, it ensures the account has not been terminated and that the specified journal type has permission to use the account. If the account is valid for use, *bgvoucher* adds the account for the current year.
3. If it does not locate the previous year's record, it validates the fund code, the function code, the object code, and the subfund code, and ensures that the combination of codes is valid in the gld_rec.
4. *Bgvoucher* sets default gla_recs as follows:
   - Blanking out the gla_rec.old_acct
   - Blanking out the gla_rec.subs
   - Defaulting all permissions to Y.
   - Defaulting summarization to N.
   - Defaulting the cash_account flag to N.
   - Defaulting the acct_terminated flag to N.
   - Defaulting the description for gla_rec.desc1 as follows:
     - Characters 1-12 contain the fund_desc.
     - Characters 13-24 contain the func_desc.
     - Characters 25-36 contain the object_desc.
   - Defaulting the description for gla_rec.desc2 as follows:
     - Characters 1-12 contain the subfund_desc.
     - Characters 13-24 contain the description for the fifth component of the account number.
     - Characters 25-36 contain the description for the sixth component of the account number.
5. If the specified journal type has permission to use the account, *bgvoucher* adds it.
6. If the account is a subsidiary control account and tent.subs_ignore is not set to Y, *bgvoucher* verifies the existence of subsidiaries for the account. If the account is not a subsidiary control account, *bgvoucher* verifies that no subsidiaries exist for the account. For more information about subsidiary validation, see *Process description for subsidiary validation* in this section.
7. *Bgvoucher* determines that there are the correct number of cash account debits and cash

account credits, according to the Entry table (ent_table).

8. *Bgvoucher* determines that the sum of all transactions is zero.

Process Description for Subsidiary Validation

When the account is a subsidiary control account, *bgvoucher* performs additional verifications, including the following:

1. Ensures that the subsidiary amount type is the same as the journal amount type (e.g., ACT, BGT).
2. Retrieves the subsidiary entry type from the Entry table, and the subsidiary code from the Subsidiary table.
3. Checks the subs_inv_or_pay value. If the value is PAY, *bgvoucher* uses the tot_used flag and the bal_used flag from the pay_tot_used and pay_bal_used codes in the Subsidiary table. If the value is INV, *bgvoucher* uses the tot_used flag and the bal_used flag from the inv_tot_used and inv_bal_used codes in the Subsidiary table.
4. Ensures that the subsidiary number in the sube_rec is valid in the id_rec.
5. Ensures that the document code in the sube_rec is valid in the doc_table.
6. Validates the allowed journal types (e.g., tdoc_ac_allow) against the journal type.
7. Ensures that the vch_rec.amt type is ACT or ENC, and that the sube_rec.amt_type = vch_rec. amt_type.
8. Ensures that subsidiary transactions exist, and rounds the amounts in the transactions if required.
9. Ensures that no balance codes exist in the subtr_rec if the subsidiary *does not* use bals.
10. Checks the bals represented by the subtr_rec.subs, subtr_rec.bal_prd, subtr_rec.bal_code, and subtr_rec.subs_no, if the subsidiary *does* use bals.

    Note: If *bgvoucher* locates the bal, and the subsidiary is closed but the tent_ignore_constr equals Y, then *bgvoucher* can post the transaction. If the subsidiary is closed but the tent_ignore_constr does not equal Y, then *bgvoucher* cannot post the transaction.
11. Ensures that the bal code in the subtr_rec exists in the subb_table, and that the subtr_rec subsidiary matches the tsubb_subs.
12. Ensures that the subs/bal_prd/amt_type combination exists in the fscl_cal_rec.
13. Compares the dates on the journal with the dates in the fscl_cal_rec to ensure the journal dates are valid.
14. Ensures that the subtr_rec does not contain values for tot_code and tot_period, if the subsidiary does not use tots.
15. Validates the tot in the subt_rec and ensures that the subt_rec is not closed, if the subsidiary does use tots.
16. Performs the following checks, if the program cannot locate the tot in the subt _rec:
    - Ensures that the subtr_rec subsidiary code matches the tsubt_subs.
    - Uses the subas_table to validate the combination of subsidiary, bal and tot.
    - Validates the tot_prd and the dates for the tot against the tot_prd and dates in the fscl_cal_rec.
    - Ensures that the tot period is completely within the bal period, or that the bal period is completely within the tot period (i.e., verifies that the bal and tot periods do not overlap).

- Ensures that the total amount in the sube equals the general ledger transaction amount.

Process Description for Posting

After *bgvoucher* completes verification, it can post transactions. After it posts a transaction, it waits for the client to obtain the status of the posting if bgv_setnotify specified POST. If signaled processing is in effect, *bgvoucher* sends a signal to the client after it finishes verifying an entry.

The following list describes the process that bgvoucher uses for the posting pass for every gle_rec.

1. *Bgvoucher* provides default information for the gle_rec as follows:
    - Obtains the vch_ref and vch_no from the current vch_rec.
    - Increments the vch_rec entry_no, then sets gle_rec.no equal to vch_rec.entry_no.
    - Initializes gle_rec.cash_amt to 0.0.
    - Initializes gle_rec.status to P.
    - Uses the current program date for the gle_rec.date.
2. If *bgvoucher* does not locate any transactions, it sets the gle_status to U and adds it to the database.
3. If *bgvoucher* does locate transactions, it looks for the account in the gla_rec. If the record is locked, *bgvoucher* tries to read it several times. If the read is unsuccessful, *bgvoucher* sends an error message to the client.
4. If the account is a summarized account, *bgvoucher* sets the gltr_stat to D. If the account is not a summarized account, *bgvoucher* sets the gltr_stat to P.
5. *Bgvoucher* sets the recon_stat to O.
6. If the transaction affects a cash account, *bgvoucher* updates the gle_cash_amt, the vch_rec.cash_dr, and the vch_rec.cash_cr.
7. *Bgvoucher* updates the vch_rec.amt_dr and vch_rec.amt_cr.
8. If the dmm_size of the next level of the ent_dml > 0, indicating that there are subsidiary entries to process, *bgvoucher* performs the following steps:
    - Locates the subsidiary account requested by the vt file record. If the suba_rec is locked, tries to read it several times. If the program cannot locate the sub_rec, locates the subs_table entry and completes the internal suba_rec by setting the following default values:
        - – suba_rec.status to 0
        - – suba_rec.date to the program date
        - – suba_rec.def_pay_terms to tsubs.def_pay_tms
        - – suba_rec.bal_enc and suba_rec.bal_act to 0.0
        - – suba_rec.entry_no to 0
        - – suba_rec.s_id and suba_rec.c_id to 0
        - – suba_rec.alt_addr_code to tsubs_rec.alt_addr_code
        - – suba_rec.credit_rating to tsubs_rec.def_cr_rating
        - – suba_rec.def_discount to tsubs_rec.def_disc
        - – suba_rec.desc to blank
        - – suba_rec.letter to tsubs_rec.def_dunning_let
        - – suba_rec.int_waived to tsubs__rec.int_wvd
        - – suba_rec.written_off to N
        - – suba_rec.agency to N
        - – suba_rec.cust_no to blank
        - – suba_rec.auto_bank to N
        - – suba_rec.bank_code and suba_rec.bank_acct to blank

    - suba_rec.prenotif_cmpl to N
    - suba_rec.prenotif_proc to N
    - suba_rec.ofc_for_chk to tsubs_rec.ofc_for_ck
    - suba_rec.id to 0
    - suba_rec.c_serial to 0
  - Adds the suba_rec
  - Completes the sube_rec fields by setting the following values:
    - sube_rec.status to P
    - sube_rec.date to prog_date
    - table_type to ent_inv_or_pay
    - sube_rec.vch_ref to vch_rec.ref
    - sube_rec.vch_no to vch_rec.no
    - sube_rec.gle_no to gle_no
    - sube_rec.amt to 0.0
  - Adds all the subtr_rec.amts to the sube_rec.amt

9. *Bgvoucher* creates due to/from entries.
10. *Bgvoucher* adds the entry to the database.
11. *Bgvoucher* processes all the transactions for this entry as follows:
    - Adds the gltr to the database.
    - Attempts to locate the glamt_rec for the account. If the record is located and found to be locked (and does not become unlocked during repeated attempts to access the record), *bgvoucher* returns an error. If no record is located, *bgvoucher* adds a new glamt_rec to the database.
    - Adds the transaction to the proper period in the glamt_rec.
    - Updates the glamt_rec in the database, then unlocks the glamt_rec.
12. *Bgvoucher* processes subsidiaries, if required. For more information about processing subsidiaries, see *Process Description for Posting Subsidiaries* in this section.
13. *Bgvoucher* updates the in-memory vch_rec to the current time and date, and updates the current vch_rec in the database to include the time, date, and last entry number.

Process Description for Posting Subsidiaries

Posting subsidiaries requires additional processing. When *bgvoucher* encounters a subsidiary transaction, it performs the following steps:

1. Locates and locks the suba_rec for this subs and subs_no.
2. Increments the suba_rec.entry_no.
3. Sets sube_rec.no to suba_rec.entry_no.
4. Adds sube_rec to the database. If *bgvoucher* encounters a “duplicate value for primary index” error, it increments the sube_no and attempts to add the record again. If *bgvoucher* encounters the same error, it increments the sube_no up to three more times before generating an error message to the client. This unlikely condition could result from a system failure that causes the sube_no to be incorrect. If it can add the record, *bgvoucher* updates the suba_rec.entry_no.
5. Performs the following processing for all subsidiary transactions:
   - Adds the subsidiary transaction to the database.
   - For amount types of ENC, performs the following:
     - Adds the transaction to the suba_rec.bal_enc.
     - For transactions with bal codes, finds the internal subb_rec that corresponds to the transaction. If the balance specified is not currently in memory, *bgvoucher*

zeroes out the totals and adds a subb_rec to the current list in memory, and adds the transaction amount to the internal subb_rec.amt_enc.
    - For transactions with tot codes, finds the subt_rec in memory that corresponds to the total code in the transaction. If *bgvoucher* cannot locate the record in memory, it zeroes out the totals in the subt_rec. If the table type is PAY, *bgvoucher* adds the transaction amount to the subt_rec, and if the table type is INV, *bgvoucher* adds the transaction amount to the subt_rec and puts the associated amount from the transaction into the subt_rec. *Bgvoucher* then adds or updates the subt_rec to the list of subt_records in memory.
  - For amount types of ACT, performs the following:
    - Adds the transaction amount to the suba_rec.bal_act.
    - If a subsidiary balance was specified, locates the internal subb_rec that corresponds to the transaction. If the balance specified is not currently in memory, zeroes out the totals and adds a subb_rec to the current list in memory.
    - Adds the transaction amount to the internal subb_rec.amt_act.
    - For transactions with a tot code and a type of PAY, adds the transaction to the internal subt_rec.amt_pay_act.
    - For transactions with a tot code and a type of INV, adds the transaction to internal subt_rec.amt_inv_act and takes the subt_rec.assoc_amt from the subtr_rec.assoc_amt.
  - Updates and unlocks the suba_rec.

6. Performs the following processing for all subb_recs that exist in memory:
   - Finds and locks the subb_rec in the database. If *bgvoucher* cannot locate the record in the database, it adds a record with the information from the internal subb_rec. If *bgvoucher* can locate the record, it adds the totals from the internal subb_rec to the totals currently in the subb_rec and updates the database subb_rec.
   - If both the actual amount and encumbered amount are 0.0, and the fc_cls_bal flag = A (for Always Close) or if the fc_cls_bal flag is C (for Close after Closing Date) and the system date is greater than fc_closing date, then *bgvoucher* closes the balance. Under any other conditions, the status is Open.
   - Unlocks the subb_rec.
7. Performs the following processing for all internal subt_recs:
   - For the subt_recs in memory, *bgvoucher* finds and locks the subt_rec in the database. If *bgvoucher* cannot locate the record in the database, it adds a record with the information from the internal subt_rec. If *bgvoucher* can locate the record, it adds the totals from the internal subt_rec to the totals currently in the database subt_rec and updates the database subt_rec. *Bgvoucher* then unlocks the subt_rec.
8. Clears out internal lists of subsidiary balances and totals.

Subsidiary Balance and Total Processing

Bgvoucher attempts to maximize the efficiency of the process by building up an internal list of the changes that it makes to the subt and subb records, and then updating these subbs and subts only after all the transactions for this subsidiary entry have been processed. This process locks the subb_recs and subt_recs for the minimum amount of time.

Program Relationships

The following programs use *bgvoucher*:

- *acctspay*
- *approve*
- *bgtinstall*
- *billing*

- *cashier*
- *ckpost*
- *defrec*
- *docvoid*
- *filepost*
- *fixpost*
- *giftpost*
- *glclcked*
- *grvoid*
- *invdef*
- *pay*
- *purch*
- *purchasing*
- *recurent*
- *sa2sr*

Background Voucher Functions

Introduction

The *bgvoucher* functions communicate between client programs and *bgvoucher*.

The system blocks all client functions except bgv_connect, bgv_putentry, bgv_terminate and bgv_void, until *bgvoucher* has time to process and validate the call.

Functions

connect = bgv_connect <clientname>;BG_CNCT_DEF connect;
Establishes a connection to *bgvoucher*. PTP handles details, including loading and running a *bgvoucher* session, or establishing a connection to a *bgvoucher* session that is already running. The returned value is a unique positive identifier given to the client by *bgvoucher* and/or PTP that is required in all future *bgvoucher* calls. If bgv_connect returns a negative number (this should always be BG_ERR), PTP was unable to start *bgvoucher*.

status = bgv_getstat(connect, ent_id); long ent_id;
Requests a status from *bgvoucher*. If unsignaled processing is in effect, the client will then automatically wait until *bgvoucher* sends a status to it upon completion of verification or posting.
If signaled processing is in effect, *bgvoucher* will send the client the status of the entry the client specified. If ent_id = 0, *bgvoucher* will send the client the status of the first entry in its internal queue, otherwise, it will report the status of the client-specified ent_id.

Note: Possible statuses are as follows:

- BG_VERIFIED (Entry has been verified but not yet posted)
- BG_EVERIFY (Entry had an error in the verify pass)
- BG_POSTED (Entry has been successfully posted to the database)
- BG_EPOST (There was an error which prevented the entry from being posted)
- BG_POSTING (Entry is currently being posted)
- BG_VERIFYING (Entry is currently being verified)
- BG_WAITING (*Bgvoucher* is waiting for a command or entry from the client)

Status = bgv_setnotify(connect, notify_specification, &notifyvar, notifyfunc); int notify_specification; int notifyvar; intnotifyfunc();
Tells *bgvoucher* when and how to notify the client upon successful or unsuccessful completion of entry processing. The following list contains valid values for notify_specification.

Note: The first four arguments (BG_VERIFY, BG_POST, BG_SIGNAL, and BG_NOSIGNAL) can be used in multiple combinations, e.g., BG_VERIFY + BG_POST + BG_SIGNAL would set up an interrupt handler that would be called whenever an entry had completed the verification step, and again when it had finished posting the entry. Specifying BG_SIGNAL without specifying a notification variable or a notification function is not allowed.

Bgvoucher will not send an error report to the client until the client requests it. When *bgvoucher* wants to send a status to the client (as previously arranged by setnotify), it will stop processing on that connection until the client requests a status report. If a signal handling routine is set up to catch *bgvoucher* errors, *bgvoucher* will signal the client and will again stop processing that connection until the status has been retrieved.

- BG_VERIFY (Specifies that *bgvoucher* is to return the verification status for all entries)
- BG_POST (Specifies that *bgvoucher* is to return the posting status of all entries)

- BG_SIGNAL (Tells *bgvoucher* to signal the client when the status of an entry can be retrieved. If notifyfunc is non-zero, it specifies the signal handling routine; if notifyvar is nonzero, PTP will set the variable to PTP_DATA whenever there is something to be read from the connection. Either notifyvar or notifyfunc must be specified; they cannot both be given.)
- BG_NOSIGNAL (Tells *bgvoucher* to disable any current signal handling and resume unsignaled processing. BG_NOSIGNAL is the default.)

status = bgv_instruction(connect, instruction); int instruction;

Bgv_instruction sends *bgvoucher* instructions that do not require arguments. Valid instructions include the following:

- BG_ABORT (Used with the BG_WAIT command. Tells *bgvoucher* not to post the entry it stopped processing after verification.)
- BG_CONTINUE (Used with the BG_WAIT command. Tells *bgvoucher* to continue posting the entry it stopped processing after verification.)
- BG_FINISH (Finish the journal currently being processed.)
- BG_INCOMPLETE (Incomplete the journal currently being processed.)
- BG_NOCOPY (Do not save a copy of the processing in a bgfile (default).)
- BG_POST (Do both verification and posting on entries (default).)
- BG_POSTLATER (Verify entries and save them in a bgfile for later posting.)
- BG_PURGE (Deletes all entries from this connection. If *bgvoucher* is currently processing an entry, it will finish processing the entry and then delete the other entries.)
- BG_VERONLY (Do only the verification of entries, not the posting.)
- BG_WAIT (Tells *bgvoucher* to wait after verification until the client requests a status for the entry before going on to post the entry.)

Note: BG_WAIT is used when a program does not want posting to automatically occur after verification. If a program is running an AC and a PC journal, for example, and wants to post both journals only if verification of both journals was successful, it might specify BG_WAIT, wait for a successful verification from both entries, and then tell *bgvoucher* to post them with BG_CONTINUE. In Purchasing, for example, an error in either one of the entries it is processing invalidates the other entry. This option is useful if BG_VERIFY has been specified with bgv_setnotify.

If signaled processing is in effect, and BG_FINISH is specified while there are still entries to be posted, bgv_instruction will return a BGV_POSTING status, signifying that the finish could not be executed at that time. The client program should wait until all entries have been posted before doing a BG_FINISH instruction.

status = bgv_savecopy(connect, *filename*);

Saves a copy of the journal session in a bgfile. If *filename* is a null string, *bgvoucher* will automatically assign it a unique postcopy name and save the copy in it. If <filename> is not null and the file specified already exists, the old file will be replaced. If BG_POSTLATER has been selected, *bgvoucher* will automatically assign a bgfile and bgv_savecopy will return an error.

If bgv_savecopy is called during a journal, it will start saving a copy of the journal until the BG_NOCOPY instruction is specified. The only savecopy files which can be guaranteed to be understood will be those begun before a journal has been started, but files begun after a journal has been started may be useful for debugging purposes.

status = bgv_terminate(connect, vchref, vchno, user);

Voids all entries in the specified journal. The journal should be finished or incompleted first.

status = bgv_void(connect, vchref, vchno, entno, user)

Voids the specified entry in the given journal. The journal should be incompleted first.

status = bgv_start(connect, &bg_head_rec); struct bg_head_rec_type bg_head_rec;
Bgv_start tells *bgvoucher* to start a new journal. The information in the header record is checked for validity. If it passes all checks, a new journal record is added to the database and the journal table field tvch_last_issued_no is incremented by one.

status = bgv_continue(connect, vchref, vchno, user); char *vchref; long vchno;
Tells *bgvoucher* to continue an incomplete journal. BG_OK is returned if the specified journal is found in the Voucher records; an error is returned if the journal is not found or otherwise invalid.

status = bgv_putentry(connect, &ent_dml, &identifier) BG_DML_DEF(ent_dml); long ent_id;
Sends one or more entries to *bgvoucher* to be verified and posted subject to the bgv_setnotify and bgv_instruction options, If more than one entry is sent, it is assumed that they are to be taken as a packet, and the entries will all be verified and then all posted. For example, if a program sends two entries, the first will be verified, the second will be verified, then the first entry will be posted, and finally the second entry will be posted. Identifier is an identifier assigned to the entry packet by the client. *Bgvoucher* does not require it to be a unique value.

connect = bgv_output(*filename*);
Sets up a special PTP connection to a file. This bypasses *bgvoucher* completely. It is primarily envisioned as being used when *bgvoucher* cannot be loaded and the client must continue processing. It may also be of value in continuing processing after *bgvoucher* fatal errors.

status = bgv_getvch(connect, &vch_rec);
Returns information about the journal. The vch_rec includes journal number, last entry, amt_cr, amt_dr, and other information which may or may not be of interest to the client. If a journal has not been continued or started, this will return an error.

status = bgv_getlines(connect, line1, line2); char line1[81], line2[81];
Upon receiving a *bgvoucher* error, the client should always call bgv_getlines to retrieve an error message that better describes the problem. If the error which occurred will invalidate any subsequent entries, bgv_getlines is called to delete the rest of the entries from *bgvoucher's* internal connection queue. Otherwise, *bgvoucher* will automatically resume normal posting of any entries in its internal structures.

Internal Functions

The following functions are internal to *bgvoucher's* processing:

status = bgv_mkfile(*filename*, specification); char *filename*[81]; int specification;
Creates *filename* which can be used for later bgv_xxxxx routines. If specification is BG_POSTLATER, *filename* will be of the format used to create bgfiles which are to be posted after by *filepost*. If specification is BG_COPY, *filename* will be in a format that will identify the file as a copy and which will be ignored by *filepost*.

Functions Used by *bgvoucher*

Bgvoucher uses the following functions:

connect = bgv_accept(clientname) char clientname[81];

Accepts a connection from a client. The client's name substitutes for clientname in the function. If the client's name is longer than 80 characters (perhaps via an extremely long pathname), it is truncated to 80 characters. PTP automatically takes care of assigning clients to *bgvoucher* and signaling *bgvoucher* to accept new clients.

int instruction; instruction = bgv_getinst(connect);
Returns an integer code specifying one of the following *bgvoucher* instructions:

- BG_ABORT (Used in conjunction with BG_WAIT. Abort processing on this entry.)
- BG_CONTINUE (Used in conjunction with BG_WAIT. Continue and post this entry.)
- BG_FINISH (Finish the current journal.)
- BG_NOCOPY (Do not save (or stop saving) a transcript of the *bgvoucher* session.)
- BG_POST (Post and verify entries.)
- BG_POSTLATER (Verify entries, but instead of posting them immediately write it to a bgfile for later posting.)
- BG_PURGE (Deletes all entries from this connection. If *bgvoucher* is currently processing an entry, it will finish processing the entry and then delete the other entries.)
- BG_VERONLY (Only verify entries.)
- BG_GETSTAT (Instructs *bgvoucher* to give the client a status report.)
- BG_SETNOTIFY (Set up procedure used to notify the client during processing.)
- BG_SAVECOPY (Save a transcript of the current journal session.)
- BG_TERMINATE (Terminate the specified journal.)
- BG_VOID (Void the specified journal.)
- BG_START (Start a journal.)
- BG_CONTVCH (Continue a journal.)
- BG_PUTENTRY (Add the entry to *bgvoucher's* internal entry structure.)
- BG_GETLINES (Retrieve error information.)
- BG_WAIT (Wait for status from verify before going on to posting.)

status = bgv_getentry(connect, &ent_dml); BG_DML_DEF(ent_dml);
Gets a general ledger entry from the client and puts it into the entry dml.

status = bgv_getstart(connect, &bg_head_rec) struct bg_head_type bg_head_rec;
Starts a journal with the information in the header record given.

status = bgv_getcontinue(connect, vchref, &vchno, &user); char vchref[3]; long vchno;
Gets a journal reference and journal number that the client wants to continue.

status = bgv_getvoid(connect, vchref, &vchno, &vchent, &user); char vchref[3]; long vchno; int vchent;
Gets a journal reference, a journal number, and a journal entry that the client wants to void.

status = bgv_getterminate(connect, vchref, &vchno, &user); char vchref[3]; long vchno;
Gets a journal reference and journal number of a journal that the client wants to terminate.

status = bgv_putstatus(connect, ent_stat, ent_id); long ent_id;
Sends the current status of the specified entry to the client.

status = bgv_putlines(connect, errline1, errline2) char errline1[81]; char errline2[81];
Sends two error lines to the client. If no error has occurred, the string "No error has occurred" will be sent. If *bgvoucher* gets a PTP error, the PTP error message will be sent in the first error line.

status = bgv_getnotify(connect, &notify_specification) int notify_specification;
Returns the description of how and when *bgvoucher* is supposed to notify the client of the progress of the entry.

status = bgv_writestat(connect, statvar); int statvar;

Sends a *bgvoucher* status to the client’s bgv routine. This differs from bgv_putstatus in that there is no entry identifier.

SECTION 7 - BUDGET REVIEW

Overview

Introduction

This section provides reference information about the Budget Review (*bgtreview*) program. The General Ledger module uses *bgtreview* to provide users with the ability to view budgeted, actual and encumbered amounts in a variety of ways.

Program Features Detailed

This section contains details about the following features of the *bgtreview* program:

- Process flow
- Table usage
- Parameters
- Program screens

Program Files

All the program files for *bgtreview* appear in the following directories:

- $CARSPATH/src/accounting/bgtreview
- $CARSPATH/src/Lib/libacct

Tables Used in the Program

The *bgtreview* program uses the following tables and records:

fs_table
The Financial Statement table that organizes elements of the G/L account into blocks, groups, and schedules

gl_amt_rec
The General Ledger Amount record that provides summarized totals for G/L accounts over fiscal periods

gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

gle_rec
The General Ledger Journal Entry records that contain information about each entry

gltr_rec
The General Ledger Transaction records that contain the amount and account charged for each transaction in an entry

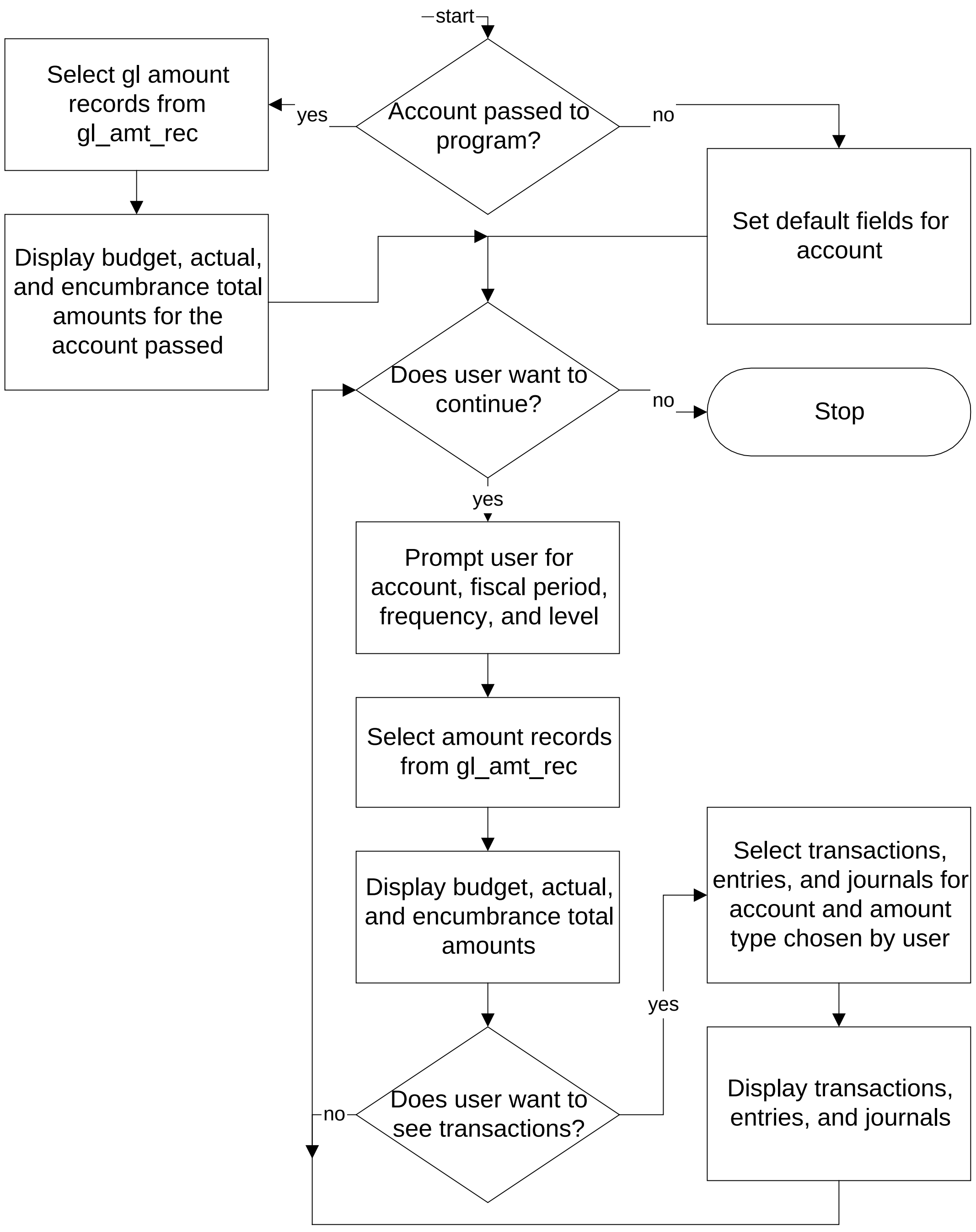
pendreq_rec
The Pending Requisition record that contains dollar amount information about submitted requisitions that have not yet been approved or ordered

vch_rec
The General Ledger Journal records that contain information about the journal

Process Flow

Diagram

The following diagram shows the flow of data in the *bgtreview* program, which relies on a library routine in src/Lib/libacct.


start
Account passed to program?
yes
Select gl amount records from gl_amt_rec
Display budget, actual, and encumbrance total amounts for the account passed
no
Set default fields for account
Does user want to continue?
no
Stop
yes
Prompt user for account, fiscal period, frequency, and level
Select amount records from gl_amt_rec
Display budget, actual, and encumbrance total amounts
Does user want to see transactions?
yes
Select transactions, entries, and journals for account and amount type chosen by user
Display transactions, entries, and journals
no

Data Flow Description

The following process describes the data flow in the *bgtreview* program.

1. If a general ledger account is passed to the program, then it does the following:
    - Selects glamt_rec for account passed
    - Displays totals of budget, actual, and encumbrance amounts

    Note: *Bgtreview* is a stand-alone program that calls the library routine *review*. If a program calls the library routine directly, it can pass an account so that the amount information appears immediately.

2. If the program does not receive a general ledger account, it sets default fields for the general ledger account.
3. If the user wants to continue, the program does the following:
    - Prompts user for account, fiscal period, frequency, and level
    - Selects General Ledger Amount records corresponding to query by user
    - Displays totals of budget, actual, and encumbrance amounts
4. If the user wants to see transactions for a budget, actual, or encumbered amount, the program does the following:
    - Selects transactions, entries, and journals for the general ledger account and amount type chosen by user
    - Displays transactions, entries, and journals

    Note: The logic flow described here actually resides in the file *review.ec* in the src/Lib/libacct directory. The program, *bgtreview*, in src/accounting/bgtreview calls this library routine.

Program Relationships

As a display-only stand-alone process, the *bgtreview* program does not interract with other programs. You can, however, access *bgtreview* from the following Accounts Payable screens:

- Purchase Order Header Entry screen
- Purchasing - Requisition Selection screen
- Accounts Payable Direct Entry screen
- Accounts Payable - Requisition Selection screen

Global Data Structures

The *bgtreview* program creates and calls the following two global data structures as needed:

dtlrvw_dmm
Contains transaction information that the program displays

review_dmm
Contains amounts that the program displays

Review Library Routine

Introduction

The *bgtreview* program uses the *review* library routine to perform the query function. Originally part of the CX program *purch*, Jenzabar has enhanced the *review* routine to be a stand-alone program.

Input to the *review* Routine

The *review* routine uses the following input:

***glacct**
The General Ledger account to be reviewed

***fscl_yr**
A character field containing the fiscal year to be reviewed (required)

mode
An integer field containing the mode to exit the review screen. Valid values are as follows:

- RVW_RET (returns to the calling program)
- RVW_BYE (exits to the UNIX shell)

***mesgbuf**
A character field containing any messages produced by the *review* routine.

***prntr**
A character field containing the name of the printer to use for output options.

Return Values

The *review* routine can return either of the following two values:

FATAL_ERR (-2)
The program routes an error message describing the cause of the problem in *mesgbuf*. If the calling mode is "RVW_RET" and this error status is returned, the calling program is responsible for redrawing the previous screen.

LOAD_ERR (-3)
If the calling mode is "RVW_RET" and this error status is returned, the calling program does not need to redraw its previous screen. The error occurred before the review screen was drawn.

Output From the *review* Routine

When the *review* routine completes, the screen is modified. The calling function is responsible for redrawing the screen when this function exits.

Budget Review Parameters

Introduction

CX contains parameters and compilation values for executing the *bgtreview* program. You can specify parameters to compile *bgtreview* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *bgtreview* program.

Parameter Syntax

You can display *bgtreview* parameters by entering the following: ***bgtreview*** **-,**

The following is the correct usage for running the *bgtreview* program from the UNIX shell:

```
bgtreview  -y fiscal year [-m fiscal period] [-f display frequency] -p output printer
```

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following lists the parameters for running *bgtreview* .

-y fiscal year
Required - The fiscal year that you want to review

-m fiscal period
Optional - The fiscal period that you want to review (e.g., JAN, BAL)

-f display frequency
Optional - The display frequency of account balances (e.g., A for all fiscal period amounts)

-p output printer
Required - The name of the printer to use in output options

Program Screens

Introduction

The *bgtreview* program uses three screens: two program screens that contain budget information and one help screen.

Access

The screen files are located in the following directory path: $CARSPATH/src/Lib/libacct/SCR

Screen Files and Table/Record Usage

The *bgtreview* screens appear in the following files and use the indicated tables and records:

review
Contains the Budget Review screen
Tables/Records: gl_amt_rec, gla_rec

rvwhelp
Contains help information for the Budget Review screen

trans
Contains the Budget Transactions screen
Tables/Records: gla_rec, gle_rec, gltr_rec, vch_rec

SECTION 8 - BURSAR QUERY

Overview

Introduction

This section provides reference information about the Bursar Query (*bursar*) program. The General Ledger module uses *bursar* to enable users to view financial information about students.

Program Features Detailed

This section contains details about the following features of the *bursar* program:

- Process flow
- Parameters
- Table usage
- Program screens

Program Files

All the program files for *bursar* appear in the following directories:

- $CARSPATH/src/accounting/bursar
- $CARSPATH/src/Lib/libbill

Program Access

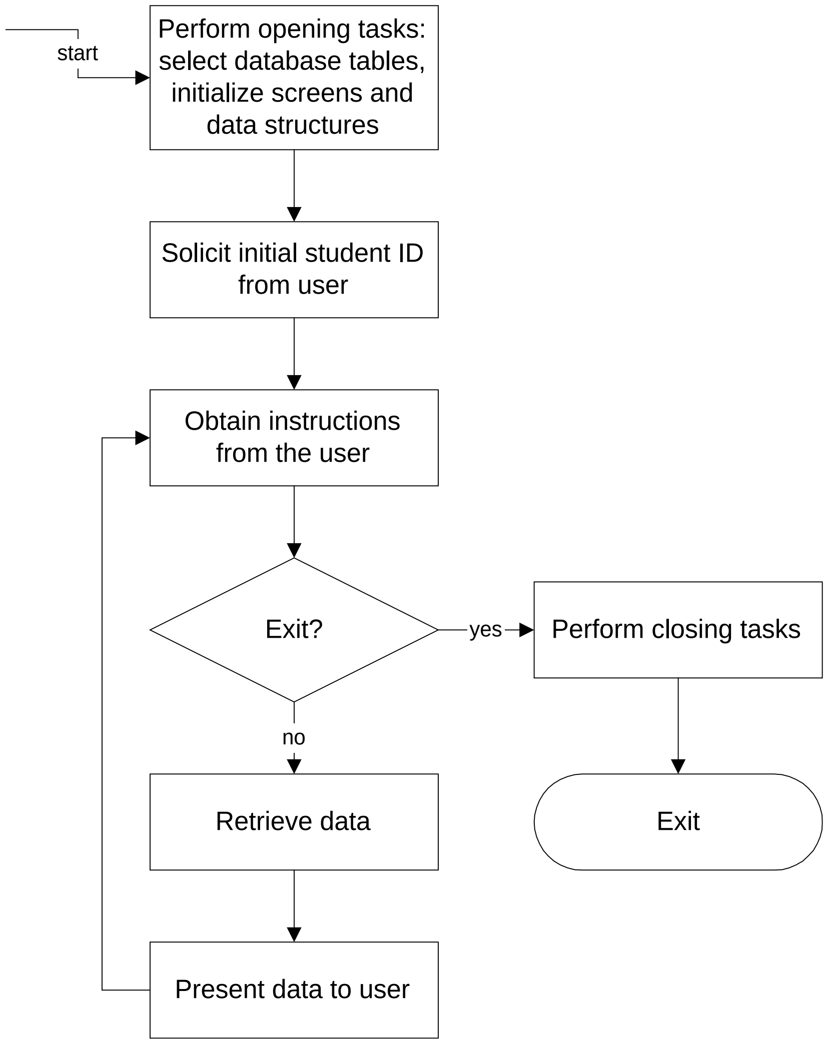
Users can access *bursar* from any of the following three menus in the CX standard product:

- Financial Management: Auditing
- Financial Management: Cash Receipts
- Financial Management: Student Billing Menu

Process Flow

Diagram

The following diagram shows the flow of data in the *bursar* program.


start
Perform opening tasks: select database tables, initialize screens and data structures
Solicit initial student ID from user
Obtain instructions from the user
Exit?
yes
Perform closing tasks
no
Retrieve data
Exit
Present data to user

Data Flow Description

The following process describes the data flow in the *bursar* program.

1. The primary program file, *main.c*, accesses *bsr.c*, a file in $CARSPATH/src/Lib/libbill.
2. Within *bsr.c*, the program calls *bill_init_bursar*, which in turn calls *bsrdb.c*, a source file with several subroutines that access the database.
3. Based on instructions from *bsrdb.c*, the program selects tables and initializes files and screens.
4. The program displays a Bursar Query screen in query mode. The default display screen is the Bursar Balance screen.
5. The user enters the ID number of the student.
6. The program retrieves the information about the student.
7. The user selects a command (e.g., Account, Balance, Parameters, Detail, or Query), and the program processes the command using the *bill_bursar* routine in $CARSPATH/src/Lib/libbill/bsr.c
8. When the user selects Exit, the program displays the menu from which Bursar Query originated.

Program Relationships

Bursar does not interact with any other CX programs

Bursar Query Parameters

Introduction

CX contains parameters and compilation values for executing the *bursar* program. You can specify parameters to compile *bursar* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *bursar* program.

Parameter Syntax

You can display *bursar* parameters by entering the following: ***bursar* -,** and then reading your electronic mail for processing messages.

[-a runcode] [-p query period] [-s query subsidiary] [-m initial mode] [-o output device] [-D] [-g]

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following lists the parameters for running *bursar*.

-r runcode
Optional - Specifies an alternate address run code

-p period
Optional - Specifies a time period for the query (e.g., JAN, CLS)

-s subsidiary
Optional - Specifies a subsidiary for the query (e.g., S/A)

-m mode
Optional - Specifies the screen that the user wants to access for the initial query (e.g., the Bursar Balance screen)

-o printer
Optional - Specifies the printer that you want to use for printing query output

Program Screens

Introduction

The *bursar* program uses five screens: four program screens that contain different views of student financial information, and a parameter window that enables users to change the search criteria for the query.

Access

The screen files are located in the following directory path:
$CARSPATH/modules/accounting/progscr/bursar

Screen Files

The *bursar* screens appear in the following files:

account
Contains the Bursar Account Information screen

balance
Contains the Bursar Balance Information screen

query
Contains the student ID information where the user enters search criteria for the query.

session
Contains the Bursar Session Information screen

stmtparam
Contains the Bursar Default Query Parameters window

Table/Record Usage

All the *bursar* screens use the following tables:

- id_rec
- stuac_rec
- suba_rec
- subb_rec
- subs_table

SECTION 9 - FILEPOST

Overview

Introduction

This section provides reference information about the File Posting (*filepost*) program. The General Ledger module uses *filepost* to post entries to the general ledger.

Program Features Detailed

This section contains details about the following features of the *filepost* program:

- Process flow
- Program interrelationships
- Table usage
- Parameters
- Program screens

Program Files

All the program files for *filepost* appear in the following directory:
$CARSPATH/src/accounting/filepost

Tables Used in the Program

The *filepost* program uses the following tables and records:

subt_table
The Subsidiary Total table that contains information about subsidiary tot codes

subs_table
The Subsidiary table that contains information about subsidiaries

vch_rec
The General Ledger Journal records that contain information about the journal

sube_rec
The Subsidiary Entry records that contain information about each entry that impacts a subsidiary

subtr_rec
The Subsidiary Transaction records that contain the subsidiary, the amount and the account charged for each transaction in an entry

In addition, *filepost* creates and calls the following four global data structures as needed:

ent_dml
The tree structure used for passing G/L transactions to *bgvoucher*

dir_dmm
The list of files displayed during interactive mode

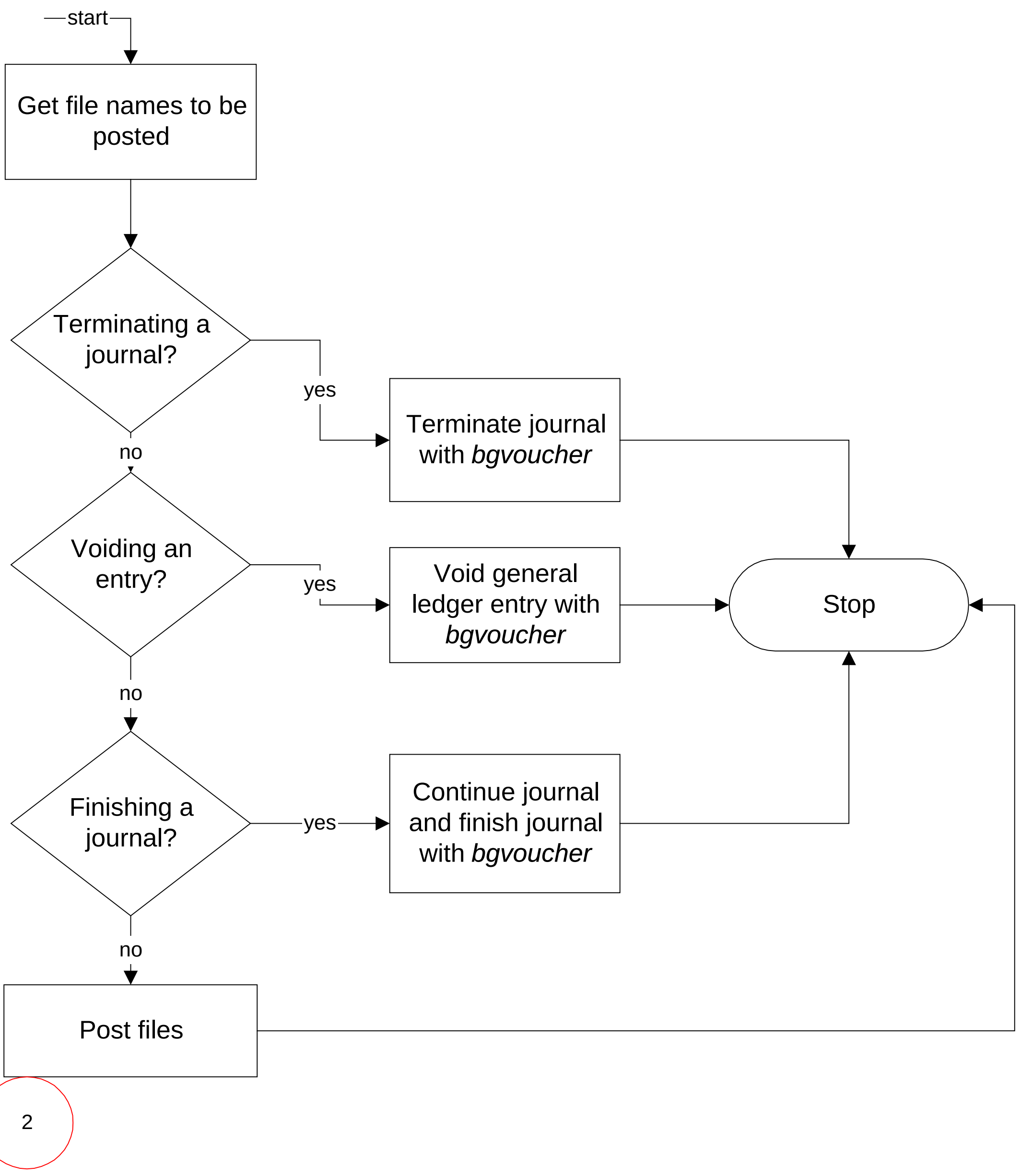
tsubs_dmm
A list of subsidiaries

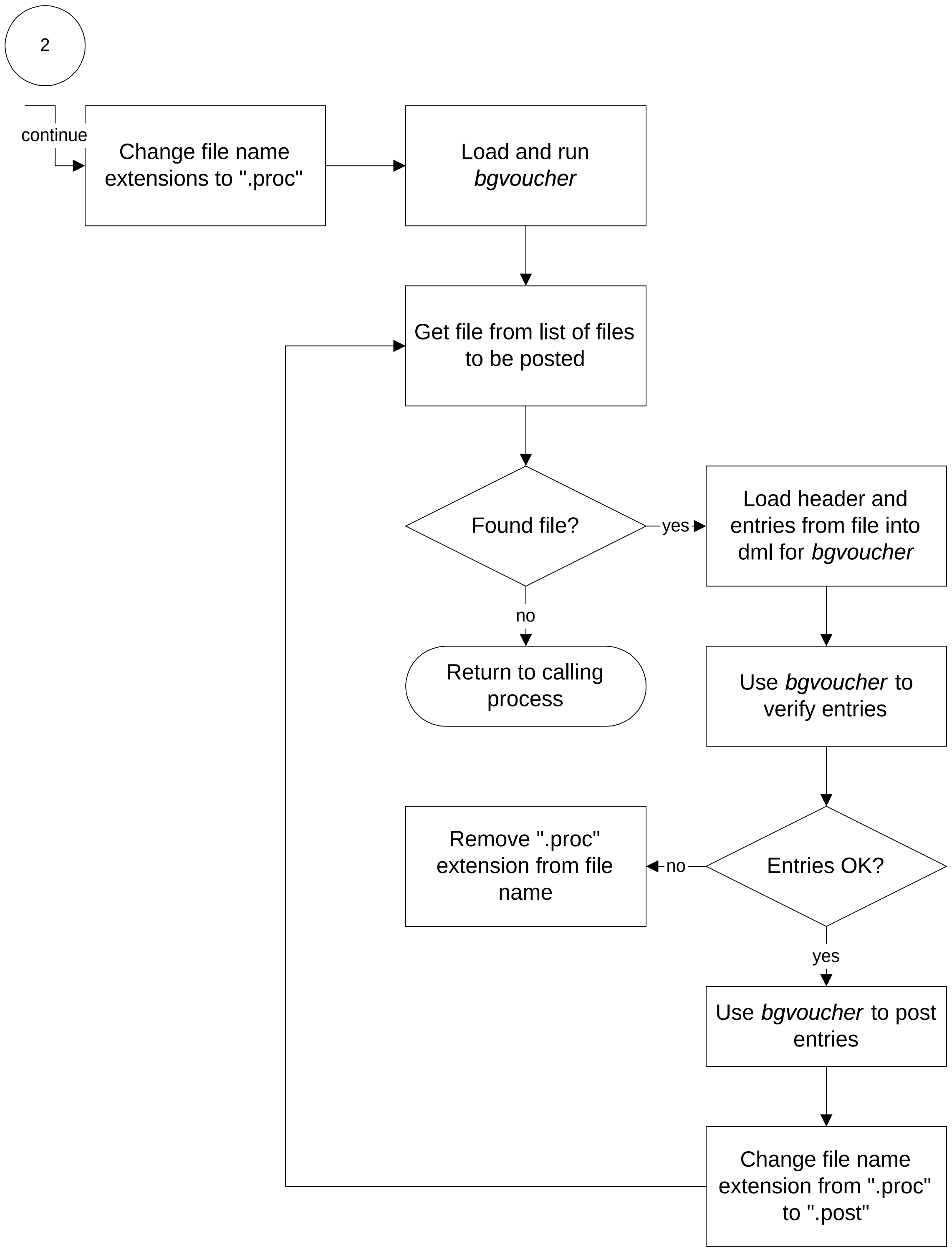
tsubt_dmm
A list of subsidiary tot codes

Process Flow

Diagram

The following diagram shows the flow of data in the *filepost* program.


start
Get file names to be posted
Terminating a journal?
yes
Terminate journal with *bgvoucher*
no
Voiding an entry?
yes
Void general ledger entry with *bgvoucher*
Stop
no
Finishing a journal?
yes
Continue journal and finish journal with *bgvoucher*
no
Post files
2


2
continue
Change file name extensions to ".proc"
Load and run *bgvoucher*
Get file from list of files to be posted
Found file?
yes
Load header and entries from file into dml for *bgvoucher*
no
Return to calling process
Use *bgvoucher* to verify entries
Entries OK?
no
Remove ".proc" extension from file name
yes
Use *bgvoucher* to post entries
Change file name extension from ".proc" to ".post"

Data Flow Description

The following process describes the data flow in the *filepost* program.

1. The user enters arguments and filenames.
2. The program parses the arguments and filenames.
3. If the user has run the program in interactive mode, the program:
   - Reads information about each file requested by the user
   - Displays file list
   - Allows the user to select files to post
   - Removes files from the dir_dmm that the user does not want to post
   - Sorts files in order of priority specified by the user
   - Goes into the background to post files
4. If the user is terminating a journal, the program:
   - Uses *bgvoucher* to terminate the journal
   - Exits
5. If the user is voiding an entry, the program:
   - Uses *bgvoucher* to void general ledger entry
   - Exits
6. If the user is finishing a journal, the program:
   - Uses *bgvoucher* to first continue the journal, then finish it
   - Exits
7. If the user uses the command line to enter files to post, the program sets defaults for the post date and period if they are not already set in each file.
8. The program performs the following steps:
   - Moves all files to ".proc" filenames
   - Loads and runs *bgvoucher*
   - Gets the first file from list, then performs the following:
     - Reads the header from the file
     - Loads General Ledger entries from the file into ent_dml
     - Uses *bgvoucher* to verify General Ledger entries before posting

   Note: If *bgvoucher* does not locate any errors, it posts the entries and renames the file from ".proc" to ".post". If *bgvoucher* locates errors, it removes the ".proc" extension from the file, and selects the next file.

File Types Used by *filepost*

Filepost can read the following three types of files:

"vt_file" type
Each record in the file has a label indicating whether it is a general ledger entry, general ledger transaction, subsidiary entry, or subsidiary transaction (old format). If the file only contains subsidiary entries, then *filepost* generates general ledger entries, and general ledger transactions.

"dml" type
Bgvoucher creates the file containing general ledger entries, general ledger transactions, subsidiary entries, and subsidiary transactions. The program loads the file into the dml via ptp.

"ascii" type

The first two characters of each line of each record indicate the type of record (e.g., a general ledger entry).

Program Relationships

The *filepost* program uses *bgvoucher* to post transactions*.*

Filepost Parameters

Introduction

CX contains parameters and compilation values for executing the *filepost* program. You can specify parameters to compile *filepost* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *filepost* program.

Parameter Syntax

The following is the correct usage for running the *filepost* program from the UNIX shell:

filepost [-d date] [-f journal reference] [-g subsidiary entry count] [-h entry count] [-I] [-m file mode] -n filenames [-p period] [-s station number] [-t terminate flag] [-v void flag]

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following list contains the parameters for running *filepost*.

-d date
Optional - Specifies the journal date, defaulting to system date.

-f journal reference
Optional - Indicates that the user wants to finish the specified journal.

-g subsidiary entry count
Optional - Indicates the number of subsidiary entries per general ledger entry.

-h entry count
Optional - Indicates that the program can use the number of entries in specified in the header as an entry count.

-I
Optional - Indicates whether the user wants to use the program in Interactive mode, defaulting to Non-interactive mode.

-m file mode
Optional - The mode of the file, defaulting to V. Valid codes include the following:

- A=ascii
- B=binary DML
- V=vt-file

-n filenames
Required - The names of files to be posted.

Note: The location of files varies with mode of file. File locations are as follows:

- A ($CARSPATH/POST_DIR/ASCII_DIR)
- B ($CARSPATH/POST_DIR/BINARY_DIR)
- V ($CARSPATH/vchpost)

-p period

Optional - The posting period (e.g., ADJ, JUN, CLS), defaulting to the period containing the current system date.

-s station number
Optional - The station number in the doc_table.

-t terminate flag
Optional - Indicates that the user wants to terminate the specified journal.

-v void flag
Optional - Indicates that the user wants to void the specified journal.

Tips for Specifying Processing Parameters

Consider the following when selecting the processing parameters to use with *filepost*:

- Finish, Terminate and Void are mutually exclusive (i.e., they cannot be specified on the same execution of *filepost*).
- Finish, Terminate and Void are not interactive options.
- If you do not use *filepost* with the Finish, Terminate or Void option, then one of the following must be true:
    - *Filepost* must be interactive (the -i flag).
    - On the command line, you must specify the name of at least one file to be posted (e.g., -n *file1 file2* ...).
- *Filepost* allows users to specify either full or partial filenames; it can locate both the exact filenames and those that match the partial filenames given on the command line.

Program Screens

Introduction

Since *filepost* works behind the scene, posting the output from other processes, it uses only one screen.

Access

The screen file is located in the following directory path:
$CARSPATH/src/accounting/filepost/SCR

Screen File

The *filepost* screen appears in the following file:

post
Provides interactive display for the user to select files to be posted

SECTION 10 - FINANCIAL STATEMENT GENERATION, FINANCIAL FORMATTING AND FINANCIAL REPORT

Overview

Introduction

This section provides reference information about the Financial Statement Generation (*fingen*), Financial Formatting (*finformat*) and Financial Report (*finrpt*) programs. These three programs work together to provide flexible reporting options that menu users can use to customize reports. The three programs together comprise the *Financial Statement Report programs*.

The report structures that you create and use from these programs provide the following features:

- Net Asset Indicators for compliance with the reporting requirements for FASB 117
- Multiple columns for comparative reporting
- Account sets for grouping accounts for greater flexibility in reporting
- Subtotaling on selected lines

Program Features Detailed

This section contains details about the following features of the *fingen*, *finformat* and *finrpt* programs:

- Process flow
- Table usage
- Parameters
- Program screens

Program Files

All program files for *fingen* appear in the following directory: $CARSPATH/src/accounting/fingen

All program files for *finformat* appear in the following directory: $CARSPATH/src/accounting/finformat

All program files for *finrpt* appear in the following directory: $CARSPATH/src/accounting/finrpt

Tables Used in the Programs

The *fingen*, *finformat* and *finrpt* programs use the following tables and records:

fin_fmt_rec
The Financial Format record that defines all the levels of the report structures on your database

fin_gl_rec
The Financial General Ledger record that links an account to a report structure level

fin_rpt_table
The Financial Report table that defines the report structure codes on your database

fin_rpt_fmt_rec
The Financial Statement Format record that establishes formats for financial statements by format code and provides columnar specifics for generating the actual report

fin_rpt_fmt_table

The Financial Statement Format table that defines valid names for report code/format code combinations

fin_set_table
The Financial Set table that defines the account groupings that you can optionally use for reporting

gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

In addition, *fingen* creates and calls the following three global data structures as needed.

bgsi_dmm
Displays information in the bgsi screen

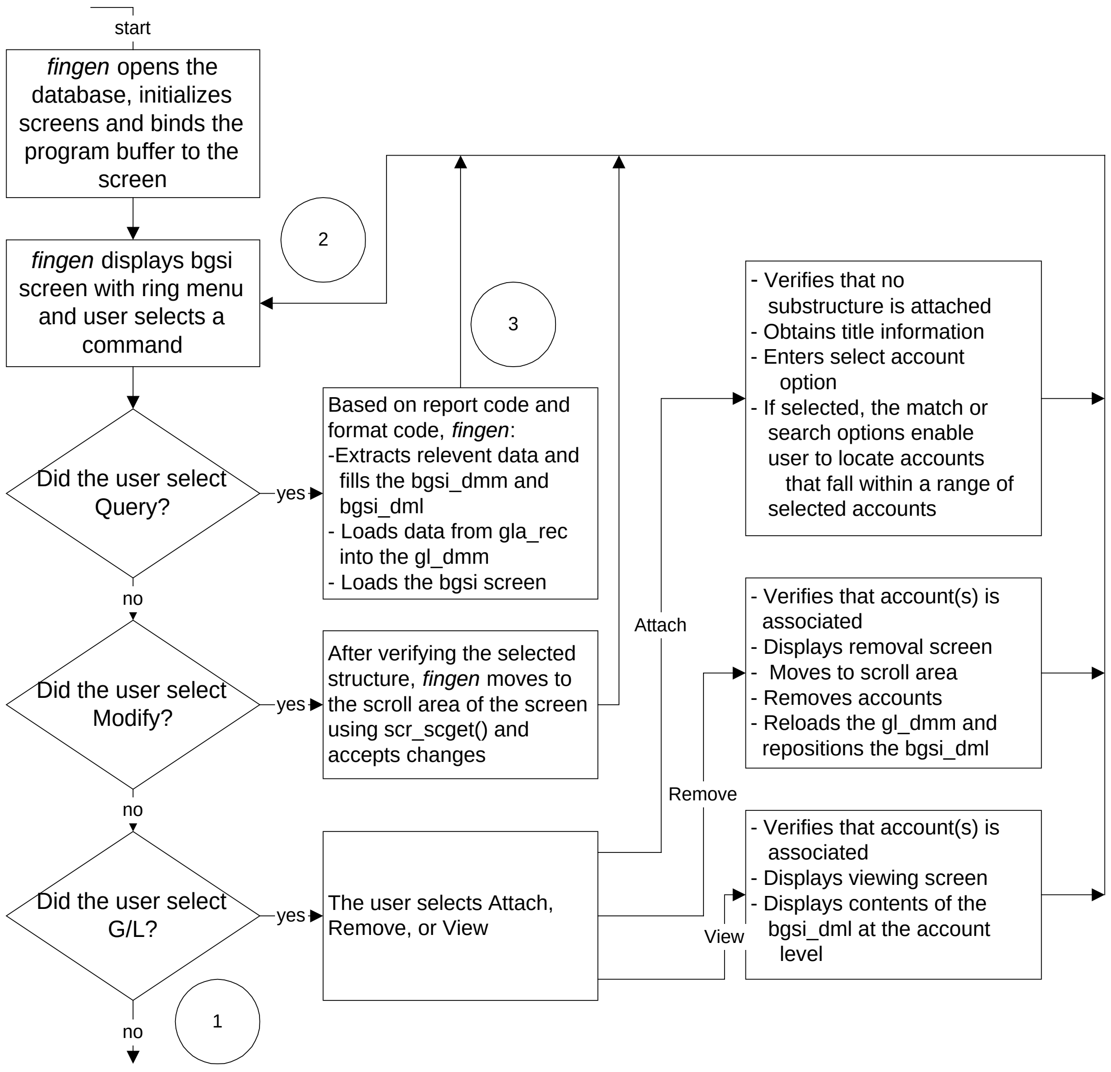
bgsi_dml
Tracks the accounts that are linked to the statement structure

gl_dmm
Contains accounts that do not exist in the fin_gl_rec for the specific report code

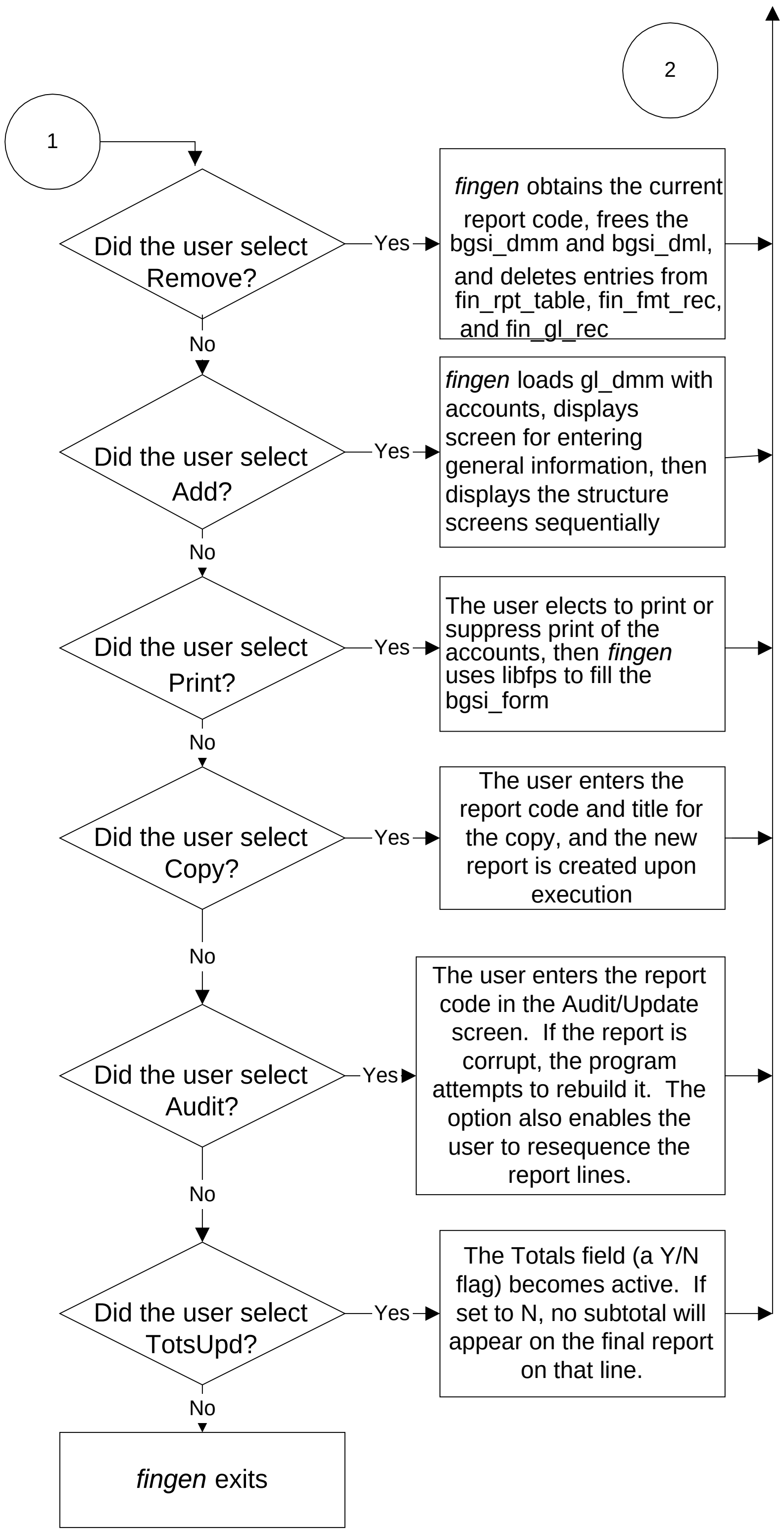
Process Flow

Diagram

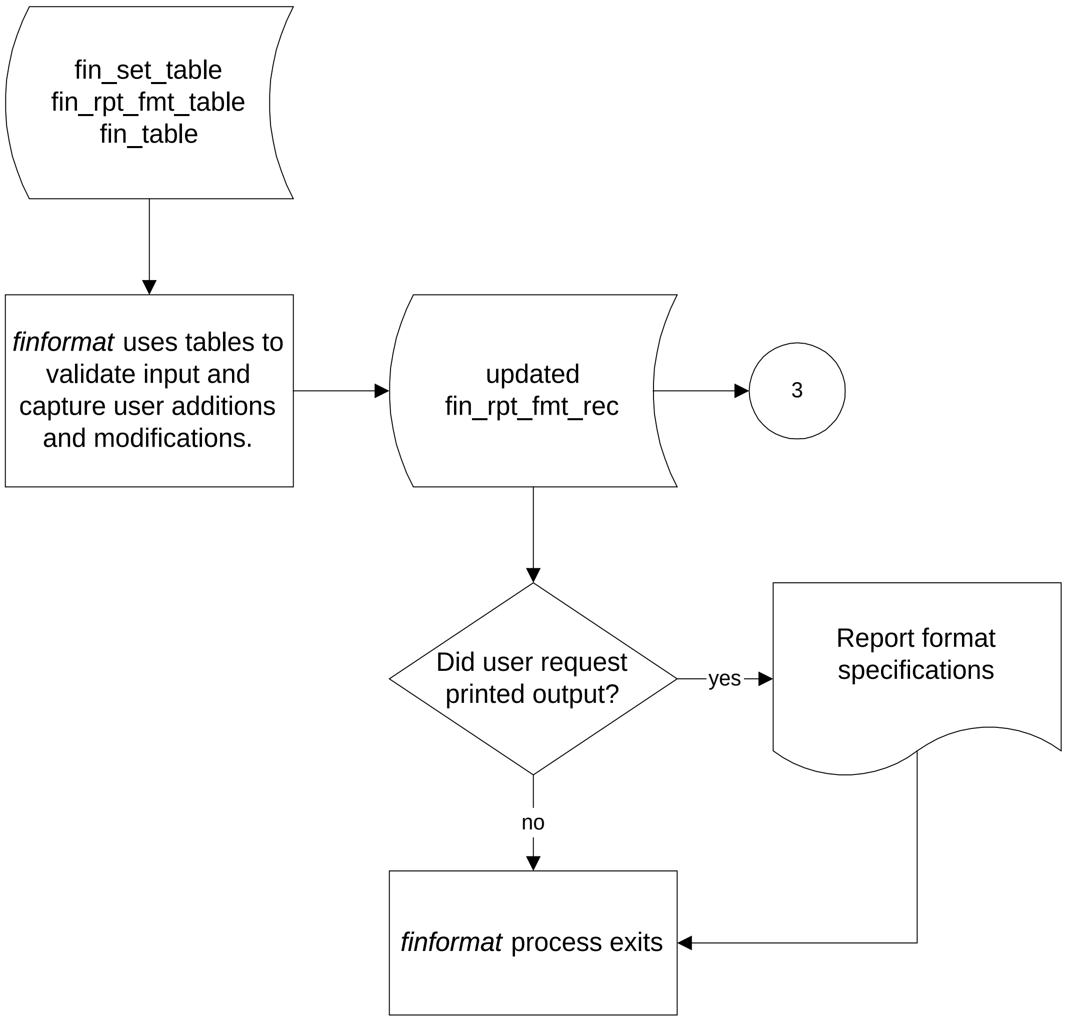
The following diagrams show the flow of data in the *fingen*, *finformat* and *finrpt* programs.



start
fingen opens the database, initializes screens and binds the program buffer to the screen
2
fingen displays bgsi screen with ring menu and user selects a command
3
Did the user select Query?
yes
Based on report code and format code, *fingen*:
-Extracts relevent data and fills the bgsi_dmm and bgsi_dml
- Loads data from gla_rec into the gl_dmm
- Loads the bgsi screen
no
Did the user select Modify?
yes
After verifying the selected structure, *fingen* moves to the scroll area of the screen using scr_scget() and accepts changes
no
Did the user select G/L?
yes
The user selects Attach, Remove, or View
no
1
Attach
Remove
View
- Verifies that no substructure is attached
- Obtains title information
- Enters select account option
- If selected, the match or search options enable user to locate accounts that fall within a range of selected accounts
- Verifies that account(s) is associated
- Displays removal screen
- Moves to scroll area
- Removes accounts
- Reloads the gl_dmm and repositions the bgsi_dml
- Verifies that account(s) is associated
- Displays viewing screen
- Displays contents of the bgsi_dml at the account level



1
2
Did the user select Remove?
Yes
fingen obtains the current report code, frees the bgsi_dmm and bgsi_dml, and deletes entries from fin_rpt_table, fin_fmt_rec, and fin_gl_rec
No
Did the user select Add?
Yes
fingen loads gl_dmm with accounts, displays screen for entering general information, then displays the structure screens sequentially
No
Did the user select Print?
Yes
The user elects to print or suppress print of the accounts, then fingen uses libfps to fill the bgsi_form
No
Did the user select Copy?
Yes
The user enters the report code and title for the copy, and the new report is created upon execution
No
Did the user select Audit?
Yes
The user enters the report code in the Audit/Update screen. If the report is corrupt, the program attempts to rebuild it. The option also enables the user to resequence the report lines.
No
Did the user select TotsUpd?
Yes
The Totals field (a Y/N flag) becomes active. If set to N, no subtotal will appear on the final report on that line.
No
fingen exits


fin_set_table
fin_rpt_fmt_table
fin_table
finformat uses tables to validate input and capture user additions and modifications.
updated fin_rpt_fmt_rec
3
Did user request printed output?
yes
Report format specifications
no
finformat process exits

Data Flow Description

The following process describes the data flow in the *fingen* program:

1. The user passes processing parameters to the program.
2. The program opens the database, initializes screens, and binds the program buffer to the screen.
3. The program displays the bgsi screen with a ring menu. Based on the ring menu selection that the user makes, the following processing occurs:

Query option

1. The user enters a report code, or selects the code using the table lookup functionality in the screen package.
2. The program extracts data from the fin_fmt_rec and the fin_gl_rec and fills the bgsi_dmm and bgsi_dml.
3. The program loads general ledger accounts from the gla_rec into the gl_dmm.
4. The program loads the bgsi screen with data from the bgsi_dmm and returns control to the ring menu.

Modify option

1. The program verifies that the user has queried a statement structure.
2. The program moves into the scroll area of the bgsi screen using scr_scget().
    - SCR_FORWARD allows the modification of the title of the current scroll line in a pop-up screen.
    - SCR_KEYU allows the insertion of additional lines into the statement structure from the current position within the scroll area.
        - The program uses the current line type and next type to create data for a selection screen which enables the user to specify what type of line is to be added.
        - A pop-up screen forces the user to make a selection of the line type for the new line.
        - Using the entered selection, the program uses the appropriate portion of the add option functionality to input data.
    - SCR_KEYV allows the removal of a branch of a statement structure, starting at the current line within the scroll area and continuing down to the general ledger account level (e.g., a schedule and its related items and accounts, or a block and its related groups, schedules, items and accounts).
        - The program uses the current line to start the deletion process.
        - The program locates the lowest level and removes it from the bgsi_dmm, the bgsi_dml, and the database. This process repeats for each successive level until the program reaches and deletes the beginning line.

G/L option

The user can select any of three options under the G/L option.

1. Attach
    - Verifies that the current line does not have any substructure lines attached to it, or that it has general ledger accounts connected directly to it.
    - Obtains the title information required by G/L Account Selection screen.
    - Enters the select account function used within the Add option.
    - Returns control to the ring menu.
2. Remove
    - Verifies that the current line has general ledger accounts connected directly to it.
    - Displays a pop-up G/L Account Removal screen.
    - Positions the bgsi_dml at the account level, and loads the screen with this data.

- Uses scr_scget() to move into the scroll area and to allow the user to indicate which accounts to remove.
- When the user completes the removal selection process, removes the selected accounts from fin_gl_rec and the bgsi_dml.
- Reloads the gl_dmm to insure that all accounts which are removed from the statement structure are available to be attached to the statement structure again.
- Repositions the bgsi_dml to the previous level.
- Returns to the Customized Financial Statement Structure screen and returns control to the ring menu.

3. View
    - Verifies that the current line has general ledger accounts connected directly to it.
    - Obtains title information needed by the viewing screen.
    - Displays a pop-up viewing screen.
    - Positions the bgsi_dml at the account level and loads the screen with this data.
    - Repositions the bgsi_dml to the previous level.
    - Returns to the Customized Financial Statement Structure screen and returns control to the ring menu.

Remove option

1. The program obtains the current report code from the bgsi_dmm.
2. The program frees bgsi_dmm and bgsi_dml and clears the screen.
3. The program deletes entries from the following tables:
    - fin_rpt_table
    - fin_fmt_rec
    - fin_gl_rec
4. The program notifies the user that the removal is complete using scr_info().
5. The program returns control to the ring menu.

Add option

The Add option operates the same as the Modify option, except upon returning to the original line type screen which was initially selected for this option. The program saves changes to the database.

1. The program loads the gl_dmm with all the general ledger accounts for the fiscal calendar that the user specified when entering the application.
2. A pop-up screen appears that enables the user to enter initial report statement data and block level data.
3. The user enters initial report statement data through scr_getset().
4. The program verifies that the report code is unique.
5. The program clears the bgsi_dml and the bgsi_dmm.
6. The program adds a blank entry to the block level of the bgsi_dml.
7. Using scr_scget() and the blank entry to bgsi_dml, the user adds the block title and the net asset indicator, and sets the G/L Next flag correctly.
8. If the user chooses SCR_ABORT, then the program returns control to the main ring menu.
9. If the user chooses SCR_DONE, then the program adds the block entry to the bgsi_dmm.

    Note: If the G/L Next flag = Y, then the program allows the user to select accounts to associate with the block, and inserts the new information into fin_fmt_rec, then goes to step 6.

10. The Group Entry screen appears, and the program adds a blank entry to the group level of the bgsi_dml.
11. Using scr_scget() and the blank entry to bgsi_dml, the user adds the group title and sets the G/L Next flag.
12. If the user chooses SCR_ABORT, then the program inserts the new information into fin_fmt_rec, and goes to step 6.
13. If the user chooses SCR_DONE, then the program adds the group entry to the bgsi_dmm.

    Note: If the G/L Next flag = Y, then the program allows the user to select accounts to associate with the group, and inserts the new information into fin_fmt_rec, then goes to step 6.
14. The Schedule Entry screen appears, and the program adds a blank entry to the schedule level of the bgsi_dml.
15. Using scr_scget() and the blank entry to bgsi_dml, the user adds the schedule title and sets the G/L Next flag.
16. If the user chooses SCR_ABORT, then the program inserts the new information into fin_fmt_rec, and goes to step 10.
17. If the user chooses SCR_DONE, then the program adds the schedule entry to the bgsi_dmm.

    Note: If the G/L Next flag = Y, then the program allows the user to select accounts to associate with the schedule, and inserts the new information into fin_fmt_rec, then goes to step 14.
18. The Item Entry screen appears, and the program adds a blank entry to the item level of the bgsi_dml.
19. Using scr_scget() and the blank entry to bgsi_dml, the user adds the item title.
20. If the user chooses SCR_ABORT, then the program inserts the new information into fin_fmt_rec, and goes to step 14.
21. If the user chooses SCR_DONE, then the program adds the item entry to the bgsi_dmm and allows the user to select accounts to associate with the item, and inserts the new information into fin_fmt_rec, then goes to step 18.

Print option

1. The user sets a flag for printing or suppressing the printing of general ledger accounts.
2. The program positions both the bgsi_dmm and the bgsi_dml at the beginning.
3. Using libfps, the bgsi_dmm and bgsi_dml fill the bgsi form.
4. If the user wants to print general ledger accounts, the program goes to the account level of the bgsi_dml and fills the form with the accounts associated with each level.

Audit option

1. The user selects a report code for audit/reconfiguration.
2. The program verifies the order and line/type fields in the fin_fmt_rec are consistent compared to the order/line_type fields in other records within the same report group/block/schedule. If inconsistent, the program attempts to repair the order fields(s).
3. The user can, if desired, change the order number to resequence lines of the report. The program automatically resequences any lines that are subordinate to the selected line.

Exit option

If the user selects the Exit option, the program exits and the menu appears.

Program Relationships

- The *fingen* program provides the ability to enter and modify statements and structures which serve as input to *finrpt*.
- The *finformat* program provides definitions for *finrpt*.

Financial Statement Report Program Parameters

Introduction

CX contains parameters and compilation values for executing the *fingen, finformat* and *finrpt* programs. You can specify parameters to compile *fingen, finformat* and *finrpt* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *fingen, finformat* and *finrpt* programs.

Parameter Syntax

You can display *fingen, finformat* and *finrpt* parameters by entering one of the following:

- f**ingen -,**
- **finforma**t **-,**
- **finrpt -,**

The following is the correct usage for running the *fingen, finformat* and *finrpt* programs from the UNIX shell:

fingen -y fiscal year -p printer

finformat -p printer device

finrpt -y fiscal year [-Y comparative fiscal year] -r report code for statement -m starting period -M ending period -A actual? -E encumbrance? -B budget? -x print G/L account exception list

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters for Financial Statement Generation

The following lists the parameters for running *fingen*.

-y year
Required - Specifies the base fiscal year for the report structure.

-p printer
Required - Specifies the printer that you want to use for printing report structures.

Parameters for Financial Format

The following lists the parameter for running *finformat*.

-p printer device
Required - Specifies the name of the printer that you want to use to produce output.

Parameters for Financial Report

The following lists the parameters for running *finrpt*.

-y year
Required - Specifies the base fiscal year for the report.

-Y comparative year
Optional - Specifies the fiscal year that you want to compare to the base year.

-r report code for statement

Required - Specifies the name of the structure.

-m starting period
Required - Specifies the beginning period that you want to show on the report (e.g., JUL.).

-M ending period
Required - Specifies the ending period that you want to show on the report (e.g., SEP).

-A actual ?
Required - Specifies, using Y or N, if you want to include actual (ACT) amounts on the report.

-E encumbrance?
Required - Specifies, using Y or N, if you want to include encumbered (ENC) amounts on the report.

-B budget?
Required - Specifies, using Y or N, if you want to include budgeted (BGT) amounts on the report.

-x print G/L exception list
Required - Indicates, using Y or N, if you want to produce a report that shows the accounts that are not included in the structure.

Program Screens

Introduction

The *fingen* program uses fifteen screens to capture and display financial report structures.

The *finformat* program uses one screen to capture and display column/format information.

The *finrpt* program does not require any screens, since it is a background process.

Access to *fingen* screens

The screen files for *fingen* are located in the following directory path:
$CARSPATH/modules/accounting/progscr/fingen

Access to *finformat* screen

The screen file for *finformat* is located in the following directory path:
$CARSPATH/modules/accounting/progscr/finformat

Financial Statement Generation Screen Files and Table/Record Usage

The *fingen* screens appear in the following files and use the indicated tables and records:

add
Contains the Block Entry screen
Tables/Records: fin_rpt_table, fin_rpt_record, fin_fmt_rec

addacct
Contains the G/L Account Selection screen
Tables/Records: fin_rpt_table, gla_rec

addgrp
Contains the Group Entry screen
Tables/Records: fin_rpt_table, fin_fmt_rec

additem
Contains the Item Entry screen
Tables/Records: fin_rpt_table, fin_fmt_rec

addschd
Contains the Schedule Entry screen
Tables/Records: fin_rpt_table, fin_fmt_rec

bgsi
Contains the Customized Financial Statement Structure screen
Tables/Records: fin_rpt_table, fin_rec

gllkp
Contains the G/L Account Search screen
Tables/Records: gla_rec

glslct
Contains the G/L Specific Selection screen
Tables/Records: fin_rpt_table, fin_set_table, gla_rec, slct_gla_rec

rmvacct

Contains the G/L Account Removal screen
Tables/Records: fin_rpt_table, gla_rec

rngacct
Contains the Manual/Range G/L Account Selection screen
Tables/Records: fin_set_table, gla_rec

select
Contains the Definition of New Line screen
Tables/Records: none

title
Contains the Change Title screen
Tables/Records: none

vwacct
Contains the G/L Account Viewing screen
Tables/Records: fin_rpt_table, gla_rec

Financial Format Screen File and Table/Record Usage

The *finformat* screen appears in the following file and uses the indicated tables and records:

main
Contains the Financial Report Format Table screen
Tables/Records: fin_rpt_fmt_rec, fin_rpt_fmt_table

SECTION 11 - GENERAL LEDGER AUDIT

Overview

Introduction

This section provides reference information about the General Ledger Audit (*glaudit*) program. The General Ledger module uses *glaudit* to resolve differences between detail and summary records. These differences can arise when a General Ledger program ends abnormally, or when system users incorrectly change or delete records using UNIX tools outside the scope of normal CX processing.

Program Features Detailed

This section contains details about the following features of the *glaudit* program:

- Process flow
- Parameters
- Table usage

Program Files

All the program files for *glaudit* appear in the following directory:
$CARSPATH/src/accounting/glaudit

Tables Used in the Program

The *glaudit* program uses the following tables and records:

atype_table
The Amount Type table that contains information about the types of amounts that you maintain on your CX database (e.g., ACT, ENC)

chrecon_rec
The Cashier Reconciliation records that contain information about the reconciliation status of General Ledger transactions

doc_table
The Document table that contains information about document codes and stations

fscl_cal_rec
The Fiscal Calendar records that define fiscal calendar years and periods

gl_amt_rec
The General Ledger Amount records that contain summarized amounts over fiscal periods

gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

gle_rec
The General Ledger Journal Entry records that contain information about each entry

gltr_rec
The General Ledger Transaction records that contain the amount and account charged for each transaction in an entry

vch_rec

The General Ledger Journal records that contain information about the journal

In addition, *glaudit* creates and calls the following six global data structures as needed:

fundbal_dmm
A list for checking fund balance within an entry

fc_tb
A table for Fiscal Calendar records

atype_tb
A table of amount types

au_ptr
A table of balances for General Ledger accounts over fiscal year

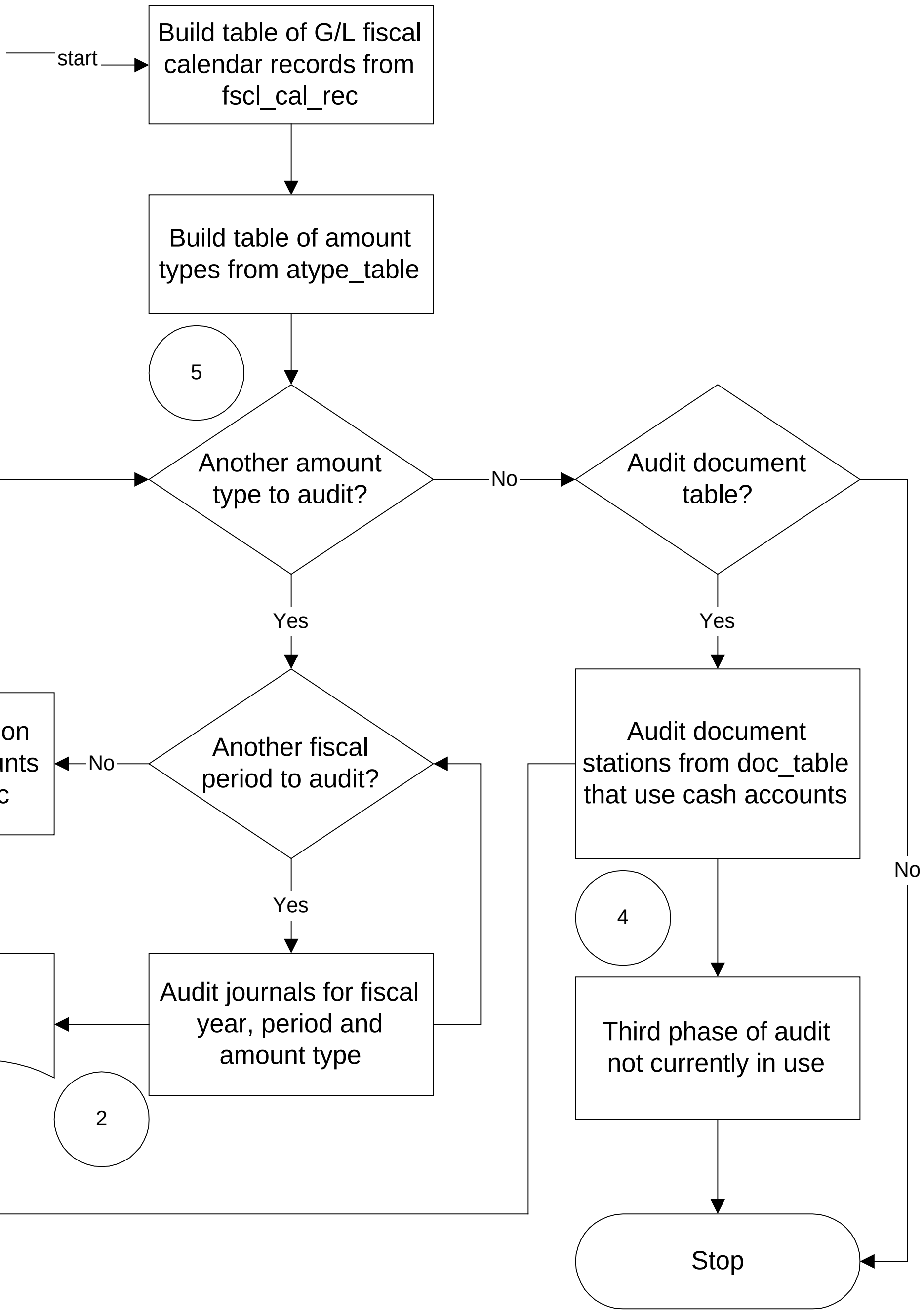
vch_dmm
A list of journals being audited

cash_dmm
A list of General Ledger cash accounts

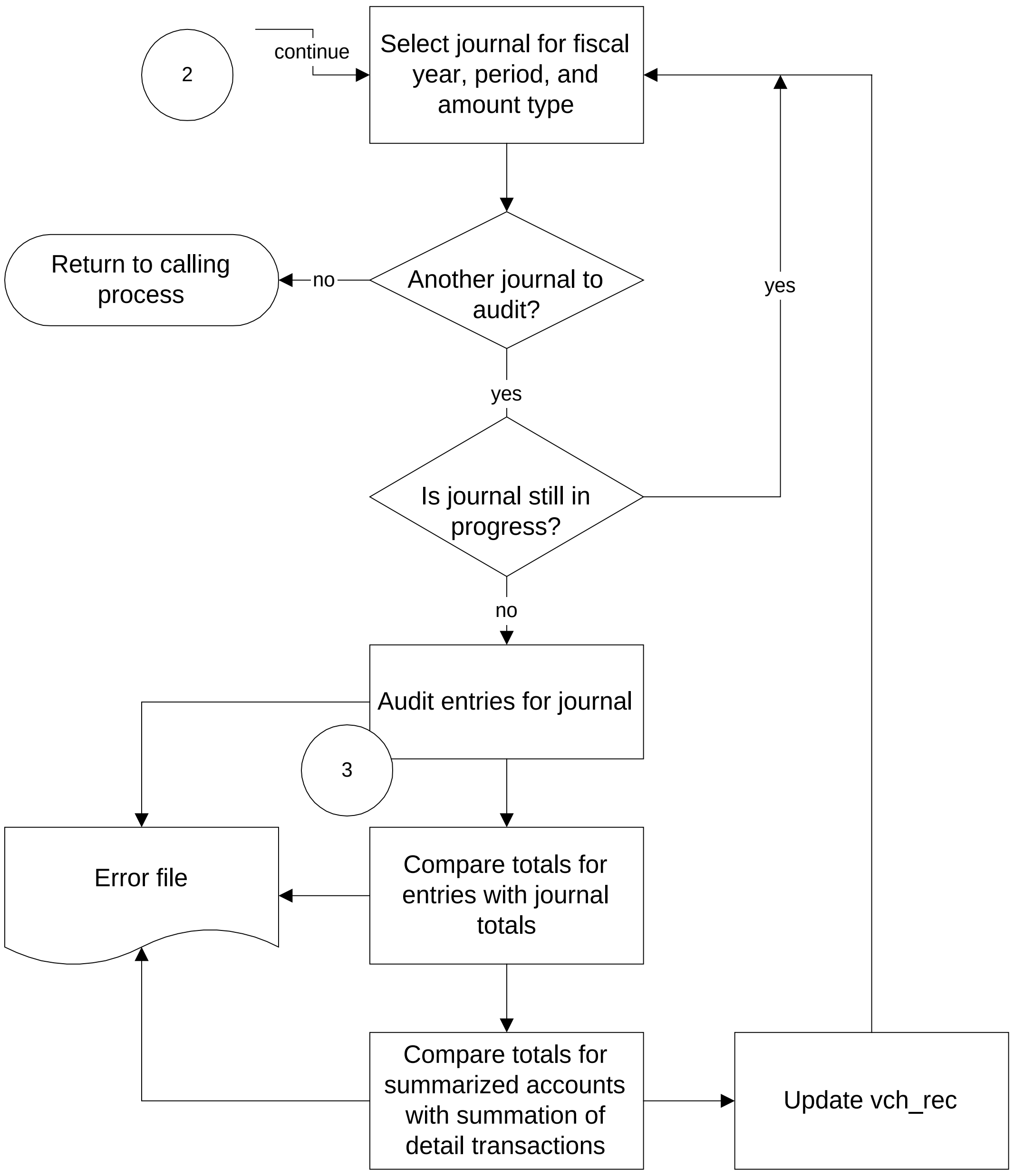
Process Flow

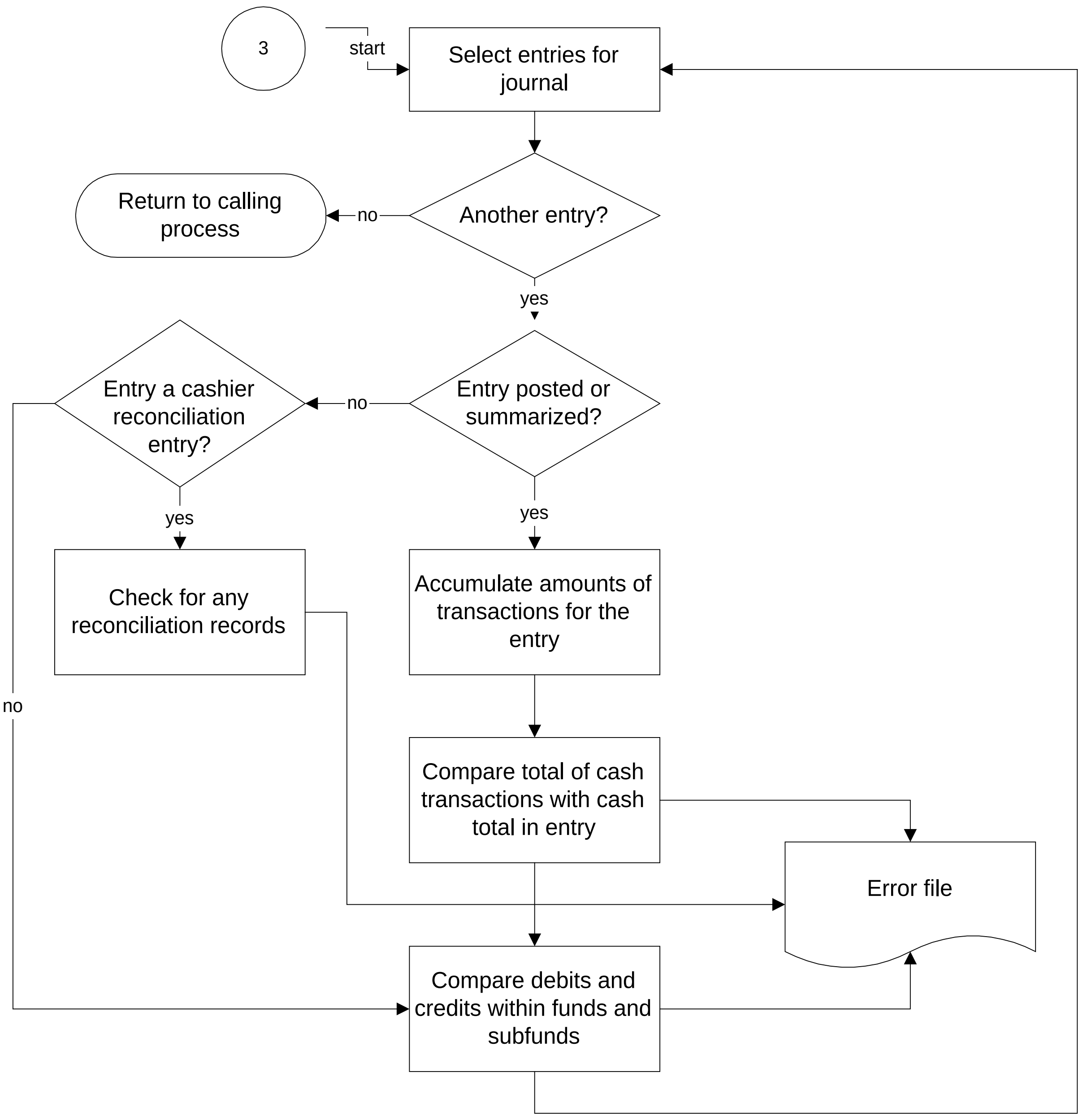
Diagram

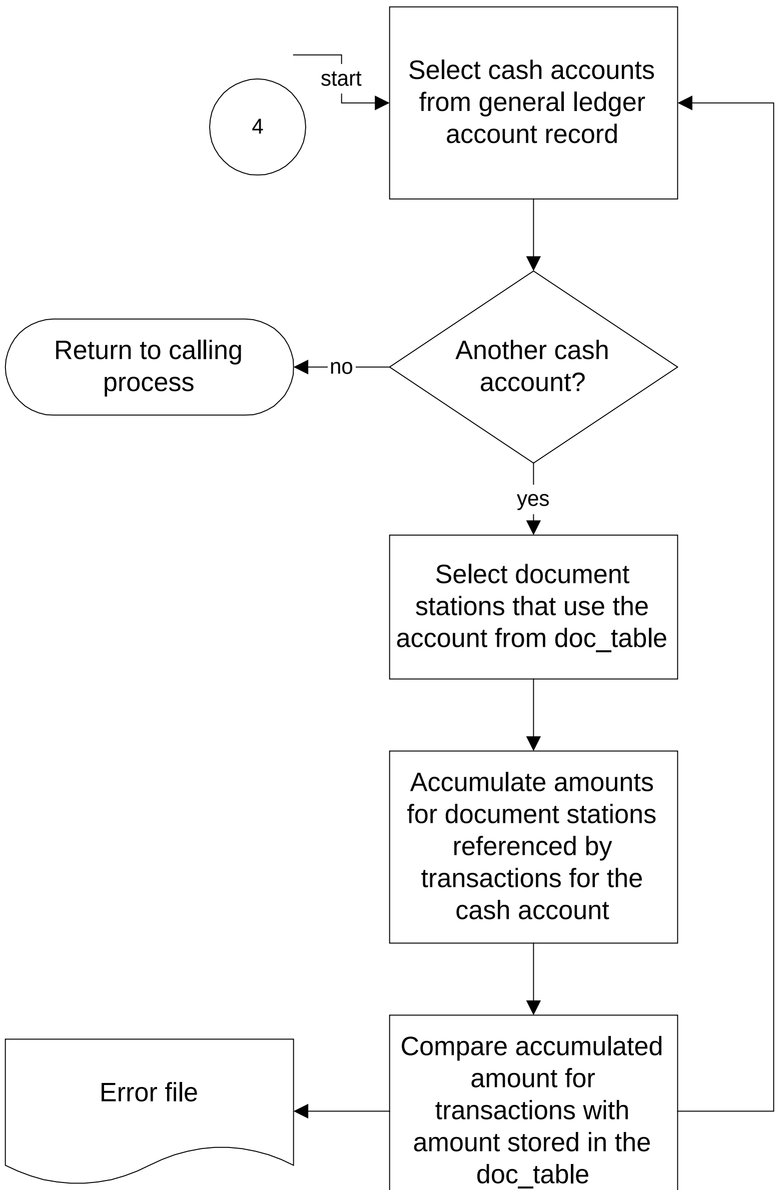
The following diagram shows the flow of data in the *glaudit* program.

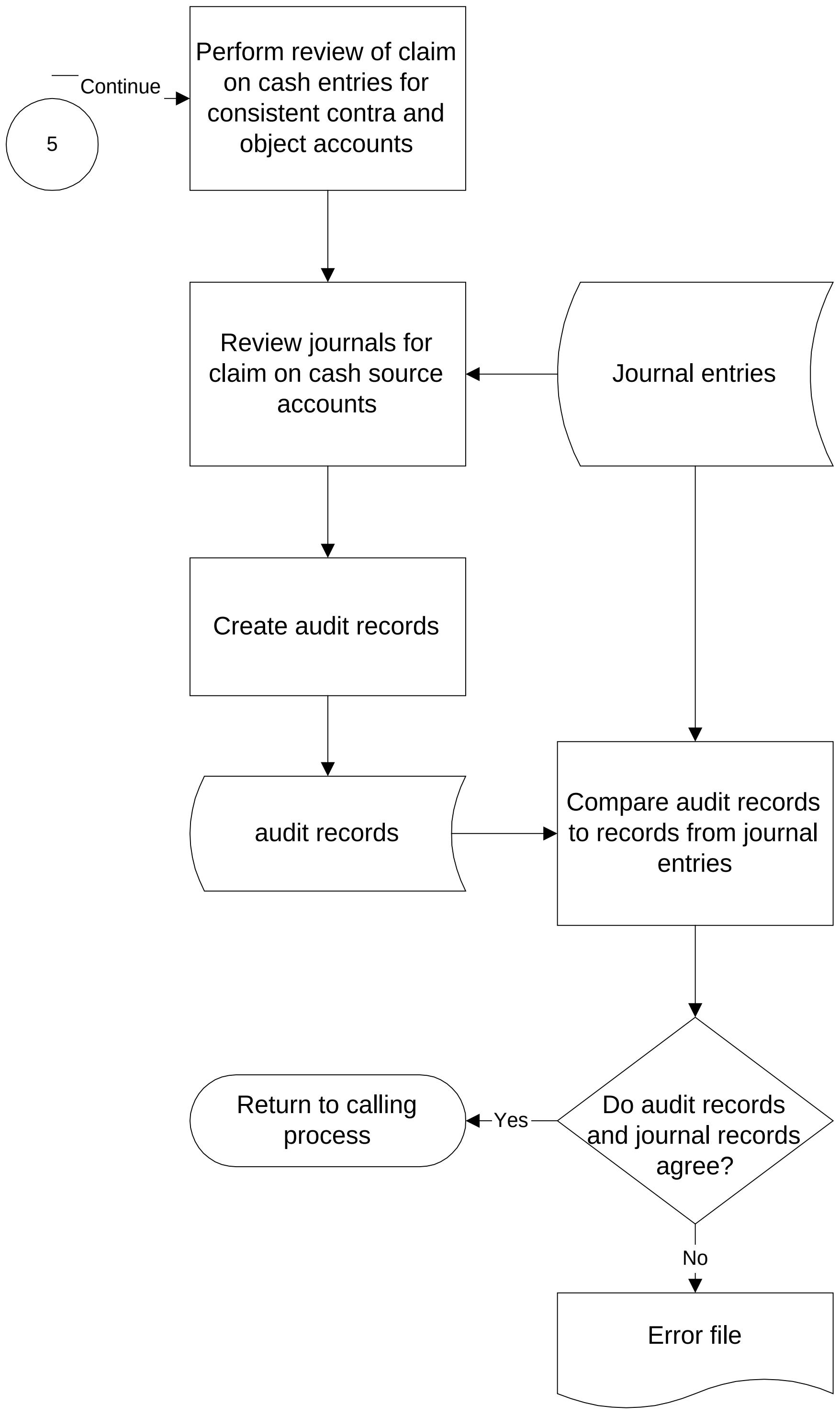


start
Build table of G/L fiscal calendar records from fscl_cal_rec
Build table of amount types from atype_table
5
Another amount type to audit?
No
Audit document table?
Yes
Yes
No
Compare transaction amounts with amounts in the gl_amt_rec
No
Another fiscal period to audit?
Audit document stations from doc_table that use cash accounts
Yes
4
Error file
Audit journals for fiscal year, period and amount type
Third phase of audit not currently in use
2
Stop


2
continue
Select journal for fiscal year, period, and amount type
Return to calling process
no
Another journal to audit?
yes
yes
Is journal still in progress?
no
Audit entries for journal
3
Error file
Compare totals for entries with journal totals
Compare totals for summarized accounts with summation of detail transactions
Update vch_rec


3
start
Select entries for journal
Another entry?
no
Return to calling process
yes
Entry posted or summarized?
no
Entry a cashier reconciliation entry?
yes
Check for any reconciliation records
yes
Accumulate amounts of transactions for the entry
Compare total of cash transactions with cash total in entry
Error file
no
Compare debits and credits within funds and subfunds


start
4
Select cash accounts from general ledger account record
Another cash account?
no
Return to calling process
yes
Select document stations that use the account from doc_table
Accumulate amounts for document stations referenced by transactions for the cash account
Compare accumulated amount for transactions with amount stored in the doc_table
Error file


5
Continue
Perform review of claim on cash entries for consistent contra and object accounts
Review journals for claim on cash source accounts
Journal entries
Create audit records
audit records
Compare audit records to records from journal entries
Do audit records and journal records agree?
Yes
Return to calling process
No
Error file

Data Flow Description

The following process describes the data flow in the *glaudit* program.

1. The program loads the cash_dmm with cash accounts from the gla_rec.
2. The program builds a table of Fiscal Calendar records.
3. The program builds a table of amount types.
4. If the institution is using the claim on cash feature, the program verifies the following:
    - If multiple table entries use the same contra account, they must also use the same object account.
    - No improperly generated claim on cash entries exist.
    - Using the journals for the specified time period, performs the following:
        - Identifies transactions that impact claim on cash processing
        - Processes the transactions against the claim_table, stripping the claim on cash contra and object accounts from the entry and finding appropriate subsidiary transaction information
        - Creates the subsidiary transaction's associated gltr_rec number if needed.
    - Compares the entries generated by the audit process to the entries in the general ledger, routing exceptions to the error file.
5. When the program identifies a new amount type to audit, it determines if there is another fiscal period to audit.
6. If it locates another fiscal period, *glaudit* selects a journal for the fiscal year/period/amount type from vch_rec.
7. When the program identifies a new journal to audit, it determines if the journal is Incomplete, Finished, or Void.
8. If the journal is Incomplete, Finished, or Void, then the program selects a general ledger entry for the journal
9. When the program identifies a general ledger entry for the journal, the program determines if the entry is Posted or Summarized.
10. If the entry is Posted or Summarized, the program does the following:
    - Accumulates amounts of general ledger transactions of the entry
    - Compares total cash amounts for transactions with cash amounts for the entry
11. If the entry is a cashier reconciliation entry, the program determines that there are no reconciliation records.
12. The program checks that all funds balance, locates the next general ledger entry, then repeats steps 8-10 as needed for subsequent entries.
13. The program checks for the following:
    - Totals from gl entries equal journal totals.
    - Summarized accounts equal the summation of the detail transactions.
14. If you are using the program in the update mode, and the journal requires an update, the program updates the vch_rec for the journal.
15. The program repeats steps 7-14 as needed for subsequent journals.
16. The program repeats steps 6-15 for subsequent fiscal periods.
17. The program compares the transaction amounts against account amounts in the gl_amt_recs.
18. The program repeats steps 5-17 for subsequent amount types.

19. If you are using the program in the Audit Document table mode, the program locates each cash account and performs the following:
    - Finds the document stations from the doc_table that use the account.
    - Accumulates the amount pertaining to the document station for each general ledger transaction for the account.
    - Reports any discrepancies between the accumulated amount for the document station and the stored amount in the doc_table record.

    Note: The *glaudit* program files contain a third phase that is currently not in use.

Program Relationships

The *glaudit* program does not interact with any other CX programs.

General Ledger Audit Parameters

Introduction

CX contains parameters and compilation values for executing the *glaudit* program. You can specify parameters to compile *glaudit* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *glaudit* program.

Parameter Syntax

You can display *glaudit* parameters by entering the following: **glaudit -,**

The following is the correct usage for running the *glaudit* program from the UNIX shell:

glaudit -y fiscal year -t type -m period [-u] [-n mail list] [-c] [-d audit document table]

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following lists the parameters for running *glaudit*.

-y fiscal year
Required Specifies the fiscal year for which you want to perform the audit.

-t type
Required - Specifies the account type that you want to audit (e.g., ACT, BGT).

-m period
Required - Specifies the time period for which you want to perform the audit (e.g, JUL-JUN).

-u
Optional - Indicates that you want *glaudit* to correct the General Ledger records as it performs the audit.

-n mail list
Optional - Specifies the name(s) of user(s) to whom *glaudit* sends mail. The mail contains the results of the audit.

-c
Optional - Indicates that you want *glaudit* to send a copy of the audit report via electronic mail.

-d audit document table
Optional - Indicates, using Y or N, if you want *glaudit* to audit cash balances maintained in the doc_table.

SECTION 12 - GENERAL LEDGER BALANCE FORWARD

Overview

Introduction

This section provides reference information about the General Ledger Balance Forward (*glbalfwd*) program. The General Ledger module uses *glbalfwd* to create beginning balances for general ledger accounts at the start of a new fiscal year, using *from* gl_amt_recs and *to* gl_amt_recs.

Program Features Detailed

This section contains details about the following features of the *glbalfwd* program:

- Process flow
- Table usage

Running Subsidiary Account Balance Forward

The Subsidiary Balance Forward process creates updated subb_recs for the current period, which are needed if your institution is using the Automated Holds feature. To ensure that Automated Holds works correctly, you must always run the Subsidiary Balance Forward option before the Automated Holds - Subs option. For more information, see the *Student Billing - Automated Holds Script* section in *Systems Manual: Student*.

Program Files

All the program files for *glbalfwd* appear in the following directory:
$CARSPATH/src/accounting/glbalfwd

Tables Used in the Program

The *glbalfwd* program uses the following tables and records:

doc_table
The Document table that contains information about document codes and stations

fscl_cal_rec
The Fiscal Calendar records that define fiscal calendar years and periods

gl_amt_rec
The General Ledger Amount record that provides summarized totals for G/L accounts over fiscal periods

vch_table
The Journal table that contains information about the valid journal types

In addition, *glbalfwd* creates and calls the following three global data structures as needed:

glamt_dmm
A list of G/L accounts to which balances are forwarded

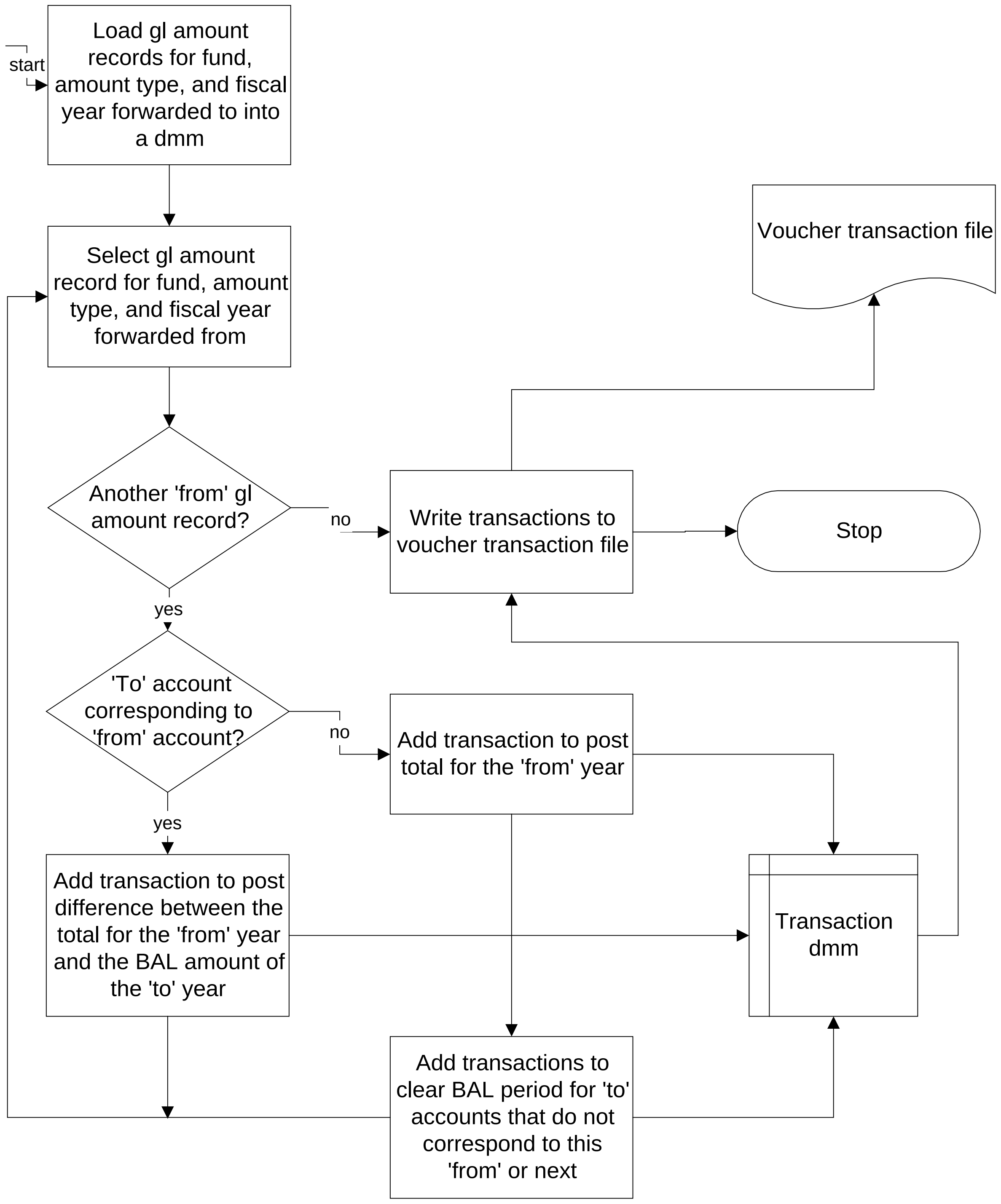
trans_dmm
A list of accounts and amounts that need to be posted

write_gt_dmm
A list of transactions to be output to vt_file

Process Flow

Diagram

The following diagram shows the flow of data in the *glbalfwd* program.


start
Load gl amount records for fund, amount type, and fiscal year forwarded to into a dmm
Select gl amount record for fund, amount type, and fiscal year forwarded from
Another 'from' gl amount record?
no
yes
Write transactions to voucher transaction file
Voucher transaction file
Stop
'To' account corresponding to 'from' account?
no
yes
Add transaction to post total for the 'from' year
Add transaction to post difference between the total for the 'from' year and the BAL amount of the 'to' year
Transaction dmm
Add transactions to clear BAL period for 'to' accounts that do not correspond to this 'from' or next

Data Flow Description

The following process describes the data flow in the *glbalfwd* program.

Note: In this process, the program accumulates existing accounting information from one year and creates beginning balances for the subsequent year. For the purposes of this data flow description, *from* data relates to the original year, and *to* data relates to the new year for which the program creates beginning balances.

1. The program builds a dmm of gl_amt_recs for the fund entered, amount type entered, and the *to* fiscal year, and stores the general ledger account and amount from BAL period.
2. The program selects the gl_amt_rec, using the following criteria:
   - The fund entered
   - The type entered
   - The originating fiscal year
3. The program checks for the following conditions. If any of the conditions are true, the program adds the total of the periods of the *from* gl_amt_rec to the list of transactions.
   - Determines if the *from* account < the *to* account
   - Determines if there are no more *to* gl_amt_recs
   - Determines if the total of the periods of the *from* gl amount record do not equal 0.0
4. If the *from* account > *to* gl amount account or there are no more *to* gl_amt_recs, the program checks if the amount in the BAL period field of the *to* gl_amt_rec does not equal 0.0. If the amount is not 0.0, the program adds a transaction to the list that causes the balance to be zero.
5. The program locates the next *to* gl_amt_rec from the dmm, and checks for the following conditions. If both of the conditions are true, the program adds the total to the list of transactions.
   - No more *to* records exist
   - The total of the periods of the *from* gl_amt_rec do not equal 0.0
6. If the *from* account < *to* account, and if the total of the periods of the *from* gl_amt_rec does not equal 0.0, the program adds the total to the list of transactions.
7. If the *from* account = *to* account, then the program does the following:
   - Checks if the total of the periods of the *from* gl_amt_rec does not equal the amount in the BAL period of the *to* gl_amt_rec. If the condition is true, the program adds a transaction to the list that adds the difference between the *from* and the *to* amount.
   - The program selects the next *to* gl_amt_rec from dmm, and the next *from* gl_amt_rec and repeats steps 2-7 above.
8. When processing from the dmm is complete, the program writes the transaction list to a file for posting.

Program Relationships

The *glbalfwd* program does not interact with any other CX programs.

General Ledger Balance Forward Parameters

Introduction

CX contains parameters and compilation values for executing the *glbalfwd* program. You can specify parameters to compile *glbalfwd* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *glbalfwd* program.

Parameter Syntax

The following is the correct usage for running the *glbalfwd* program from the UNIX shell:

-y year1 year2 -t type -f fund -v voucher

All the parameters are required.

Parameters

The following list displays the parameters for running *glbalfwd*.

-y year1 year2
Required - The year to bring forward and the new year into which to bring the balances.

-t type
Required - Amount type of account.

-f fund
Required - Fund being brought forward.

-v voucher
Required - Voucher type.

Example: An example of using the parameters for *glbalfwd* is as follows:

```
glbalfwd -y 9596 9697 -t ACT -f 10 -v AC
```

The parameters in this example brings year 9596, type ACT, fund 10 forward to 9697 with a journal type AC.

SECTION 13 - GENERAL LEDGER CLOSING

Overview

Introduction

This section provides reference information about the General Ledger Closing (*glclsg*) program. The General Ledger module uses *glclsg* to close nominal accounts (e.g., revenues and expenses) into net asset accounts, or fund balances.

Program Features Detailed

This section contains details about the following features of the *glclsg* program:

- Process flow
- Parameters
- Table usage
- Program screens

Program Files

All the program files for *glclsg* appear in the following directory:
$CARSPATH/src/accounting/glclsg

Tables Used in the Program

The *glclsg* program uses the following tables and records:

atype_table
The Amount Type table that contains information about the types of amounts that you maintain on your CX database (e.g., ACT, ENC)

clsgfb_rec
The Closing Fund records that indicate which accounts to close into other accounts

ent_table
The Entry table that contains information about valid general ledger entry types

fscl_cal_rec
The Fiscal Calendar records that define fiscal calendar years and periods

gl_amt_rec
The General Ledger Amount records that contain summarized amounts over fiscal periods

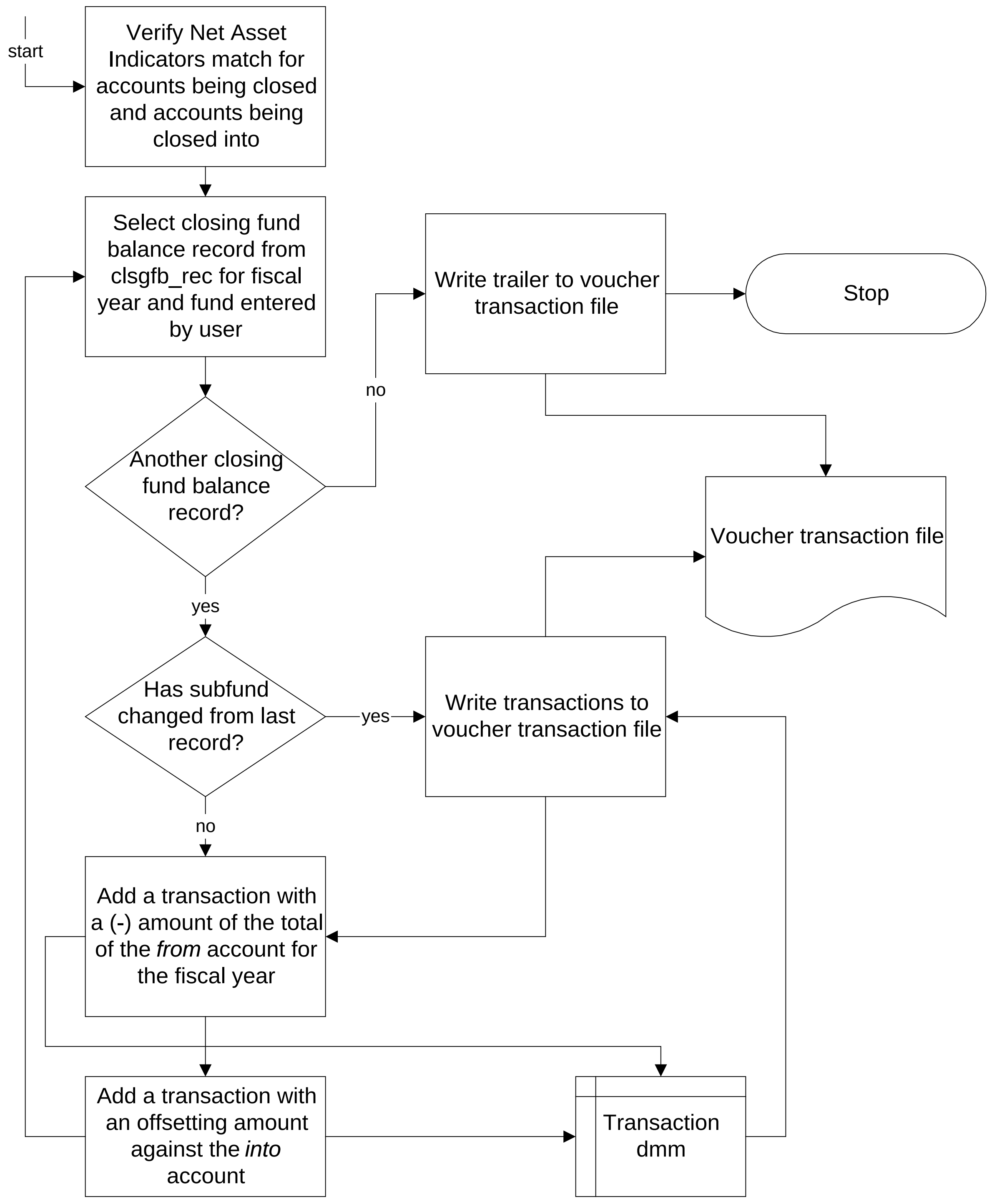
gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

vch_table
The Journal table that contains information about the valid journal types

Process Flow

Diagram

The following diagram shows the flow of data in the *glclsg* program.


start
Verify Net Asset Indicators match for accounts being closed and accounts being closed into
Select closing fund balance record from clsgfb_rec for fiscal year and fund entered by user
Another closing fund balance record?
no
Write trailer to voucher transaction file
Stop
Voucher transaction file
yes
Has subfund changed from last record?
yes
Write transactions to voucher transaction file
no
Add a transaction with a (-) amount of the total of the from account for the fiscal year
Add a transaction with an offsetting amount against the into account
Transaction dmm

Data Flow Description

The following process describes the data flow in the *glclsg* program.

1. The program performs the following verifications:
   - The amount type is valid in the atype_table
   - The amount type matches the amount type required by voucher reference
   - The fscl_cal_rec for the closing period exists and is open
   - The Net Asset Indicators in the accounts to close and the accounts into which to close are the same
2. After verification, the program selects a clsgfb_rec where the fiscal year equals the fiscal year entered by the user, and the closing-from fund equals the fund entered by the user.
3. If the closing-from subfund is different from the subfund of last record, the program reads then writes the transaction list to the voucher file.
4. The program selects the gl_amt_rec for the closing-from account.
5. The program totals the amounts for periods BAL through ADJ.
6. If the total amount does not equal the negative amount of the CLS period, the program creates a transaction against closing-from account with an amount equal to the negative of total amount for BAL through CLS.
7. The program creates an offsetting transaction against the closing-into account and retrieves the next closing fund balance record.
8. The program repeats steps 1-7 until all records have been processed.
9. The program writes the last list of transactions for a subfund to the voucher file.
10. The program writes a trailer record for the voucher file.

Program Relationships

Glclsg creates a voucher transaction (.vt) file that passes to *filepost*.

General Ledger Closing Parameters

Introduction

CX contains parameters and compilation values for executing the *glclsg* program. You can specify parameters to compile *glclsg* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *glclsg* program.

Parameter Syntax

You can display *glclsg* parameters by entering the following: **glclsg -,**

The following is the correct usage for running the *glclsg* program from the UNIX shell:

glclsg -y fiscal year -t amount type -f fund -v voucher

All the parameters are required.

Parameters

The following lists the parameters for running *glclsg*.

-y fiscal year
Required - The fiscal year that you want to close.

-t amount type
Required - The amount type that you want to close (e.g., ACT, BGT).

-f fund
Required - The fund code that you want to close (e.g., 10).

-v voucher
Required - The journal type that you want to use for the closing entries (e.g., AC).

SECTION 14 - GENERAL LEDGER CLOSING CHECK

Overview

Introduction

This section provides reference information about the General Ledger Closing Check (*glclcked*) program. The General Ledger module uses *glclcked* to ensure that the user has not attempted to close an account with a Net Asset Indicator into an account with a different Net Asset Indicator. The Net Asset Indicator provides the means by which institutions can group accounts in compliance with FASB (Financial Accounting Standards Board) Statement 117. The FASB statement requires institutions to group accounts as follows:

- Permanently restricted
- Temporarily restricted
- Unrestricted

Program Features Detailed

This section contains details about the following features of the *glclcked* program:

- Process flow
- Parameters
- Program screens

Program Files

All program files for *glclcked* appear in the following directory:
$CARSPATH/src/accounting/glclcked

Tables Used in the Program

The glclcked program uses the following tables and records:

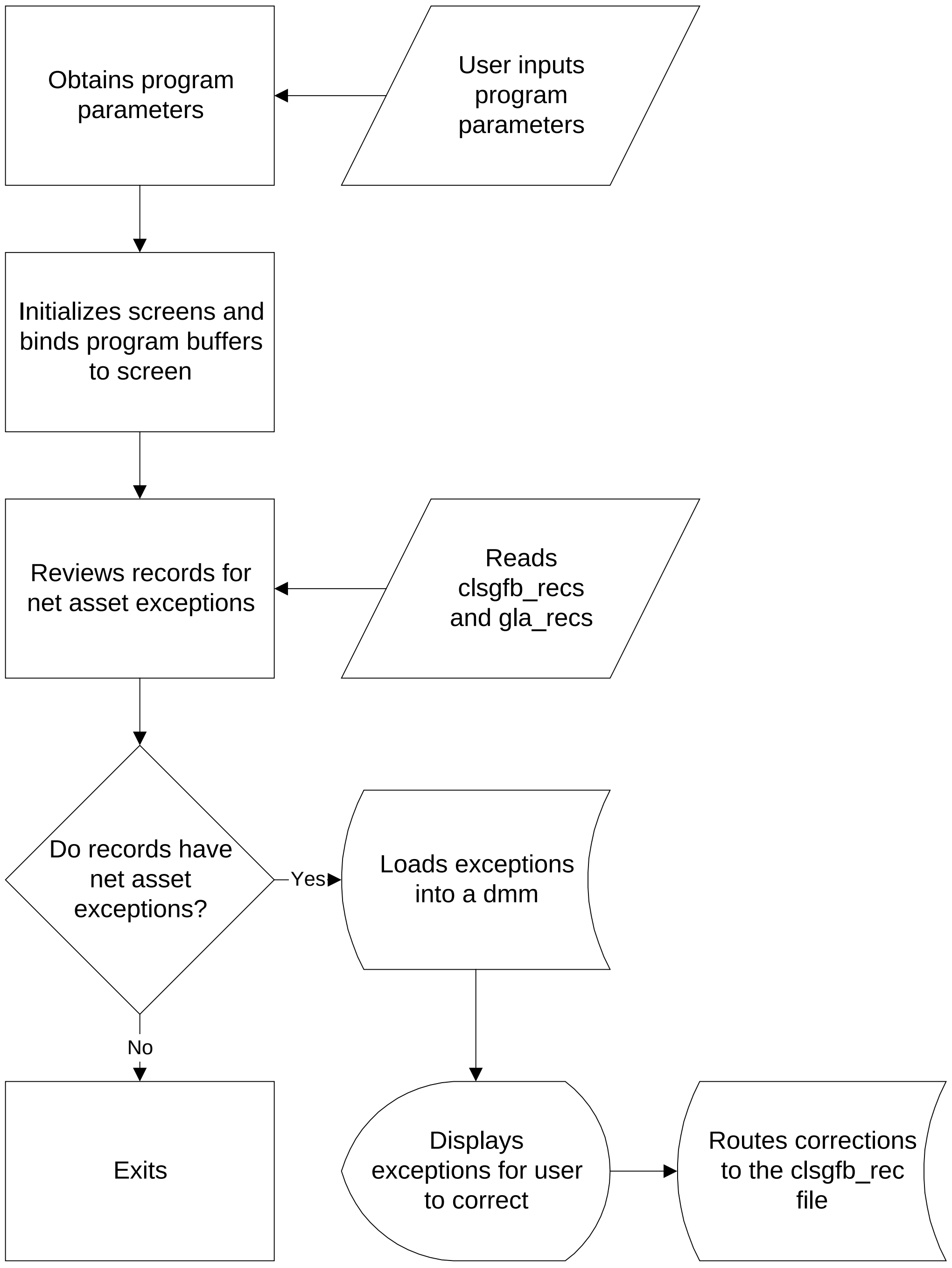
clsgfb_rec
The Closing Fund records that indicate which accounts to close into other accounts

gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

Process Flow

Diagram

The following diagram shows the flow of data in the *glclcked* program.


Obtains program parameters
User inputs program parameters
Initializes screens and binds program buffers to screen
Reviews records for net asset exceptions
Reads clsgfb_recs and gla_recs
Do records have net asset exceptions?
Yes
Loads exceptions into a dmm
No
Exits
Displays exceptions for user to correct
Routes corrections to the clsgfb_rec file

Data Flow Description

The following process describes the data flow in the *glclcked* program.

1. The user passes processing parameters to the program.
2. The program initializes screens and binds the program buffer to the screens.
3. The program accesses the clsgfb_recs and the gla_recs.
4. The program locates net asset exceptions (those accounts that the user attempted to close into an account with a different Net Asset Indicator) and routes the exceptions to a dmm. Exceptions appear on the Edit Checking for Closing Program screen.
5. The user corrects the closing entries on the Edit Checking for Closing Program screen, specifying the correct account into which to make closing entries. The corrections change the clsgfb_recs.

   Note: The user cannot change the Net Asset Indicator on an account at this time. To change Net Asset Indicators, users must access the gla_recs for the accounts.

Program Relationships

The *glclcked* program does not interact with any other CX programs.

General Ledger Closing Check Parameters

Introduction

CX contains parameters for executing the *glclcked* program. You can specify parameters to execute *glclcked* in a specified manner.

Parameter Syntax

You can display *glclcked* parameters by entering the following: **glclcked -,**

The following is the correct usage for running the glclcked program from the UNIX shell:

glclcked -f fund account -y fiscal year

All the parameters are required.

Parameters

The following lists the parameters for running *glclcked*.

-f fund account
Required - Specifies the fund number.

-y fiscal year
Required - Specifies a fiscal year (e.g., 9596).

Program Screens

Purpose

The Edit Checking for Closing Program screen enables you to view and correct closing entries that include accounts with different Net Asset Indicators.

Access

The screen file is located in the following directory path:
$CARSPATH/modules/accounting/progscr/glclcked

Screen Location in the Program

You can access the Edit Checking for Closing Program screen from the General Ledger Closing menu, or by entering at the Csh prompt: **glclcked -f fund account -y fiscal year**

Screen Files and Table/Record Usage

The *glclcked* screen appears in the following file and uses the indicated tables and records:

edit
Contains the Edit Checking for Closing Program screen
Tables/Records: clsgfb_rec, gla_rec

SECTION 15 - RECURRING ENTRY

Overview

Introduction

This section provides reference information about the Recurring Entry (*recurent*) program. The General Ledger module uses *recurent* to create and maintain repetitive journal entries. Users define repetitive entries in recur_table, and can modify the dollar amounts or the accounts in the entries and then post them as required.

Program Features Detailed

This section contains details about the following features of the *recurent* program:

- Process flow
- Parameters
- Table usage
- Program screens

Program Files

All the program files for *recurent* appear in the following directory:
$CARSPATH/src/accounting/recurent

Tables Used in the Program

The *recurent* program uses the following tables and records:

ent_table
The Entry table that contains information about valid general ledger entry types

fscl_cal_rec
The Fiscal Calendar records that define fiscal calendar years and periods

gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

glsub_table
The General Ledger Substitution table that allows you to enable the General Ledger Account Auto-Fill feature.

recur_rec
The Recur records that contain general ledger account transactions for a recur_code

recur_table
The Recur table that contains information about the valid recur_codes, accounts and amounts

subs_table
The Subsidiary table that contains information about valid subsidiaries

vch_rec
The General Ledger Journal records that contain information about the journal

vch_table
The Journal table that contains information about the valid journal types

In addition, *recurent* creates and calls the following two global data structures as needed:

recur_dmm
A list of transactions for the currently selected recur_code

org_recur_dmm
A list of transactions for a recur_code before the user makes any changes

lst_recur_dmm
A list of transactions for a recur_code before the user made the most recent change

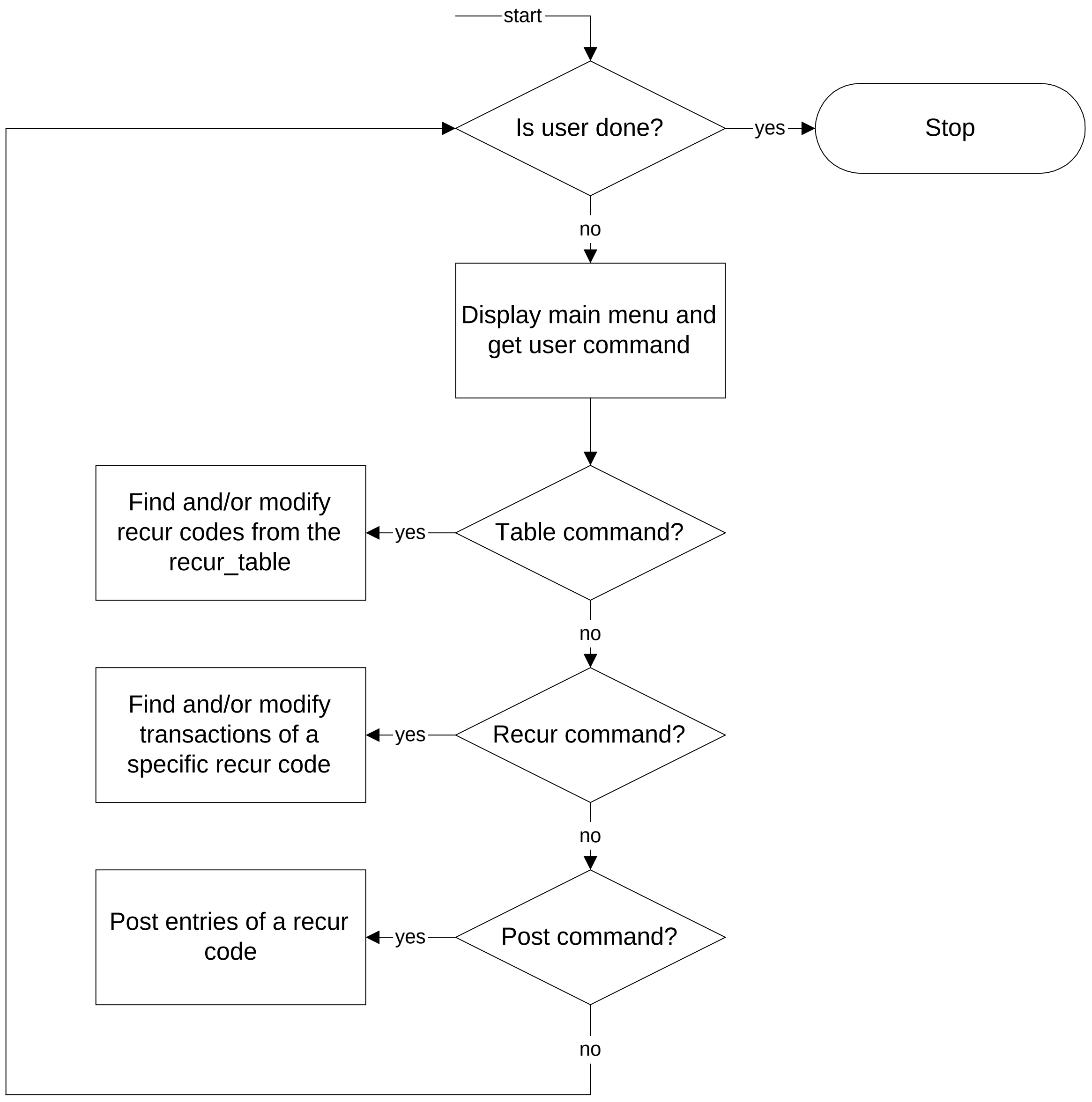
trecur_dmm
Transaction information for the current recur_code

trcur_dmm
A list of information on recur_codes

Process Flow

Diagram

The following diagram shows the flow of data in the *recurent* program.


start
Is user done?
yes
Stop
no
Display main menu and get user command
Table command?
yes
Find and/or modify recur codes from the recur_table
no
Recur command?
yes
Find and/or modify transactions of a specific recur code
no
Post command?
yes
Post entries of a recur code
no

Data Flow Description

The following process describes the data flow in the *recurent* program.

1. The user enters a journal reference type.
2. The program locates vch_table information for the journal reference type entered by user.
3. The program locates the fiscal month and period for the current date.
4. The program displays the main menu.
5. If the user enters the Table command, the program enables the user to find and/or modify the contents of recur_table.
6. If the user enters the Recur command, the program enables the user to find and/or modify transactions of a specific recur code.
7. If the user enters the Update command, the program enables the user to modify transactions of the last queried recur code.
8. If the user enters the Post command, the program enables the user to post entries of a recur code.
9. If the user enters the Exit command, the user exits from the program.

Program Relationships

Recurent creates entries that *bgvoucher* posts to the General Ledger.

Recurring Entry Parameters

Introduction

CX contains parameters and compilation values for executing the *recurent* program. You can specify parameters to compile *recurent* in a specified manner at the time of execution.

> **Note:** You can also specify compilation values with the includes for the General Ledger module that affect the *recurent* program.

Parameter Syntax

You can display *recurent* parameters by entering the following: **recurent -,**

The following is the correct usage for running the *recurent* program from the UNIX shell:

recurent [-t] [-a] [-p] [-r journal reference]

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

> **Note:** The menuopt usually sets the parameters for processing *recurent*.

Parameters

The following lists the parameters for running *recurent*.

-t
Optional - Enables the user to execute the Table command in *recurent*.

-a
Optional - Enables the user to modify or add entries for a recur code.

-p
Optional - Enables the user to post entries.

-r journal reference
Optional - Specifies the journal type to which you want to post recurring entries (e.g., AC).

Program Screens

Introduction

The *recurent* program uses two screens: one screen for the entry of table information, and the other for viewing and maintaining entries.

Access

The screen files are located in the following directory path:
$CARSPATH/modules/accounting/progscr/recurent

Screen Files and Table/Record Usage

The *recurent* screens appear in the following files and use the indicated tables and records:

recur
Contains the Recurring Entry screen
Tables/Records: gla_rec, recur_rec, recur_table

table
Contains the Recurring Entry Table screen
Tables/Records: recur_table

SECTION 16 - STANDARD ACCOUNTING ENTRIES

Overview

Introduction

This section provides reference information about the Standard Accounting Entries (*sae*) program. The General Ledger module uses *sae* to create commonly used journal entries in which users perform percentage allocations of amounts.

Program Features Detailed

This section contains details about the following features of the *sae* program:

- Process flow
- Parameters
- Program screens

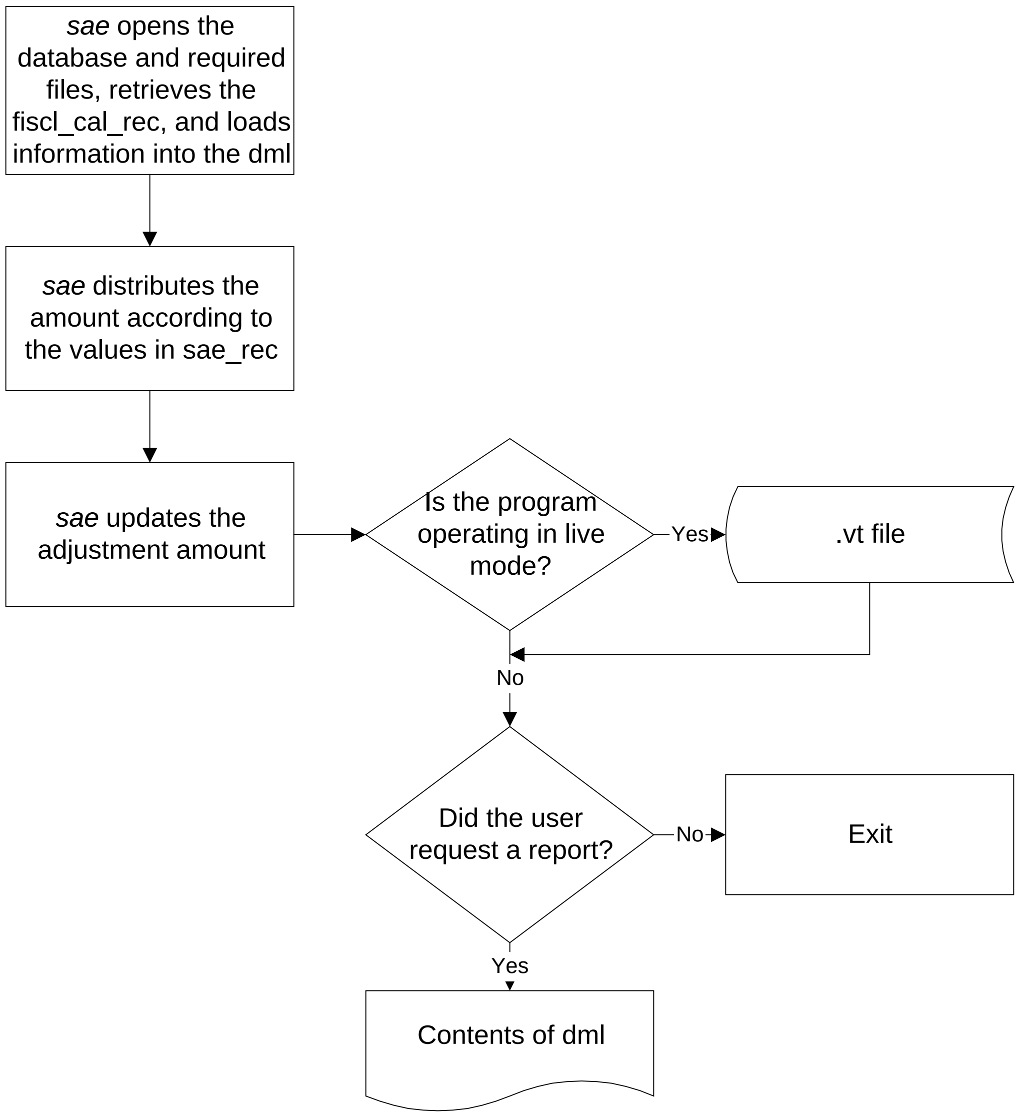
Program Files

All program files for *sae* appear in the following directory: $CARSPATH/src/accounting/sae

Process Flow

Diagram

The following diagram shows the flow of data in the *sae* program.


sae opens the database and required files, retrieves the fiscl_cal_rec, and loads information into the dml
sae distributes the amount according to the values in sae_rec
sae updates the adjustment amount
Is the program operating in live mode?
Yes
.vt file
No
Did the user request a report?
No
Exit
Yes
Contents of dml

Data Flow Description

The following process describes the data flow in the *sae* program.

1. The program obtains processing parameters from the user.
2. The program opens the database and the required files.
3. The program retrieves the fiscl_cal_rec.
4. The program loads the following information into the dml, in the order shown:
   - sae codes
   - sae_recs
   - glamt_recs
5. The program ensures that the record combinations are valid.
6. The program distributes the amount according to the values in the sae_rec.
7. The program updates the adjustment amount by performing the following calculation:
   adjustment amount = computed amount - current amount in the glamt_rec
8. If the user requested that the program produce a report, it prints the contents of the dml.
9. If the user requested that the program operate in live mode (not just report or test mode), it creates a .vt (voucher) file that the *filepost* program can post.

Program Relationships

The *sae* program creates a voucher file that *filepost* uses for posting.

Standard Accounting Entries Parameters

Introduction

CX contains parameters and compilation values for executing the *sae* program. You can specify parameters to compile *sae* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *sae* program.

Parameter Syntax

You can display *sae* parameters by entering the following: **sae -,**

The following is the correct usage for running the *sae* program from the UNIX shell:

```
sae -y year -m month [-p post date] [-d effective date] [-c sae code] [-t] [-r]
```

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following table lists the parameters for running *sae*.

-y year
Required - Specifies the year for which you want to process standard accounting entries

-m month
Required - Specifies the month for which you want to process standard accounting entries

-p post date
Optional - Specifies the posting date that you want to use for the standard accounting entries

-d effective date
Optional - Specifies the date that you want to use to locate the standard accounting entries

-c sae code
Optional - Specifies the sae codes that you want to process

-t
Optional - Specifies that you want to test the standard accounting entries without creating a vt output file for posting

-r
Optional - Specifies that you want to produce a report of the standard accounting entries created by the process

Program Screens

Introduction

The *sae* program uses one screen for creating entries to the sae_table and record.

Access

The screen file is located in the following directory path:
$CARSPATH/modules/accounting/screens

Screen File and Table/Record Usage

The *sae* screen appears in the following file and uses the indicated tables and records:

sae
Contains the screen you use to enter standard transactions.
Tables/Records: sae_rec, sae_table

SECTION 17 - SUBSIDIARY ACCOUNT BALANCE FORWARD

Overview

Introduction

This section provides reference information about the Subsidiary Account Balance Forward (*sabalfwd*) program. The General Ledger module uses *sabalfwd* to consolidate subsidiary transactions and to establish balances for subsidiary accounts for a new fiscal year.

Program Features Detailed

This section contains details about the following features of the *sabalfwd* program:

- Process flow
- Parameters
- Table usage

Program Files

All the program files for *sabalfwd* appear in the following directory:
$CARSPATH/src/accounting/sabalfwd

Tables Used in the Program

The *sabalfwd* program uses the following tables and records:

doc_table
The Document table that contains information about document codes and stations

ent_table
The Entry table that contains information about the valid entry types (e.g., ADJ for adjusting entries)

fscl_cal_rec
The Fiscal Calendar records that define fiscal calendar years and periods

subas_table
The Subsidiary Association table that contains information about the relationships between balance and total codes

subb_rec
The Subsidiary Balance records that contain information about a subsidiary balance transaction

subb_table
The Subsidiary Balance table that defines valid balance codes

subs_table
The Subsidiary table that defines valid subsidiary codes

subt_table
The Subsidiary Total table that defines valid total codes

In addition, *sabalfwd* creates and calls the following five global data structures as needed:

cr_dmm
A list of credit transactions to be written to voucher transaction file

dr_dmm

A list of debit transactions to be written to voucher transaction file

bal_dmm
A list of subb_recs for a subsidiary

from_dmm
A list of subsidiary periods to be brought forward

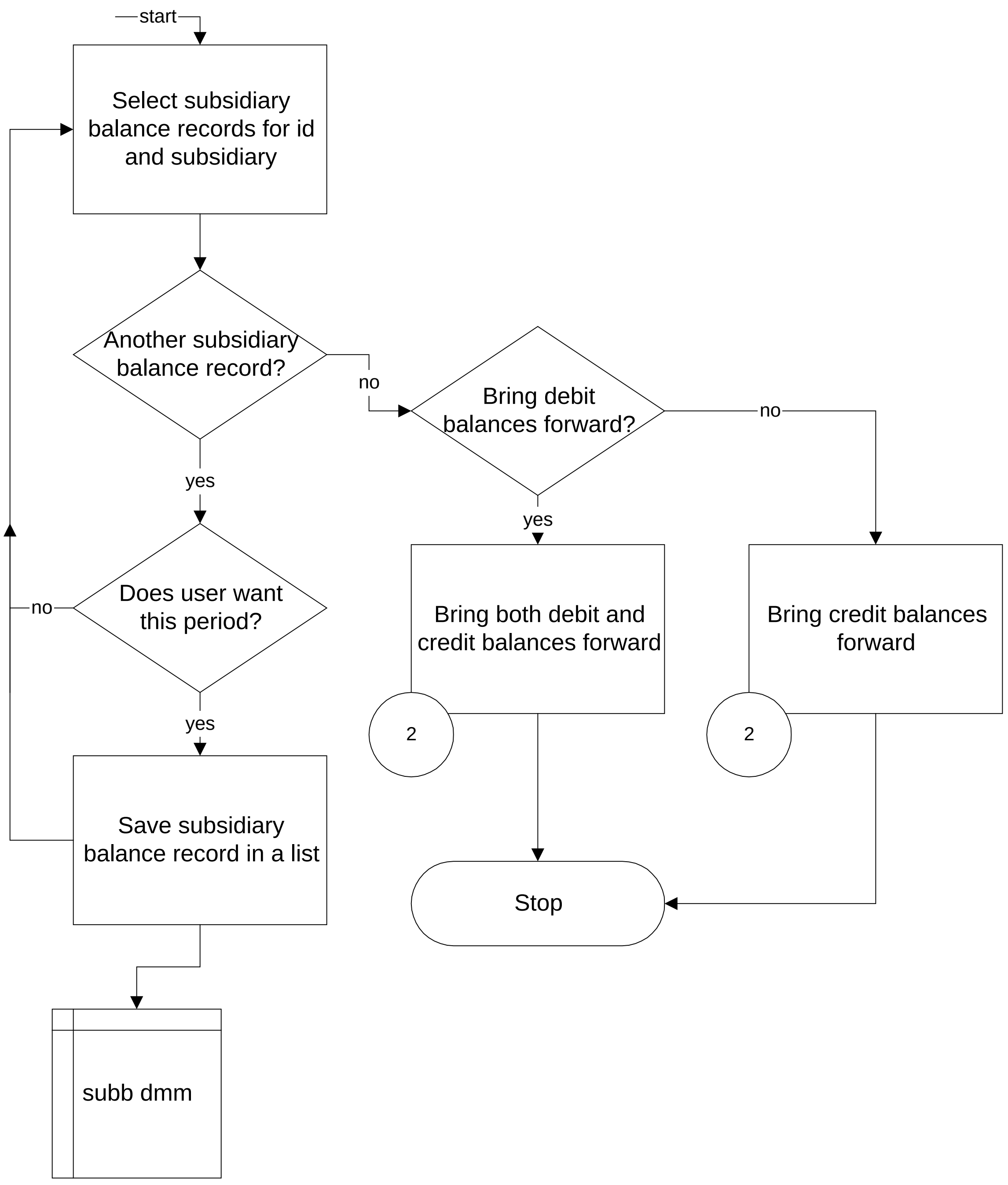
id_dmm

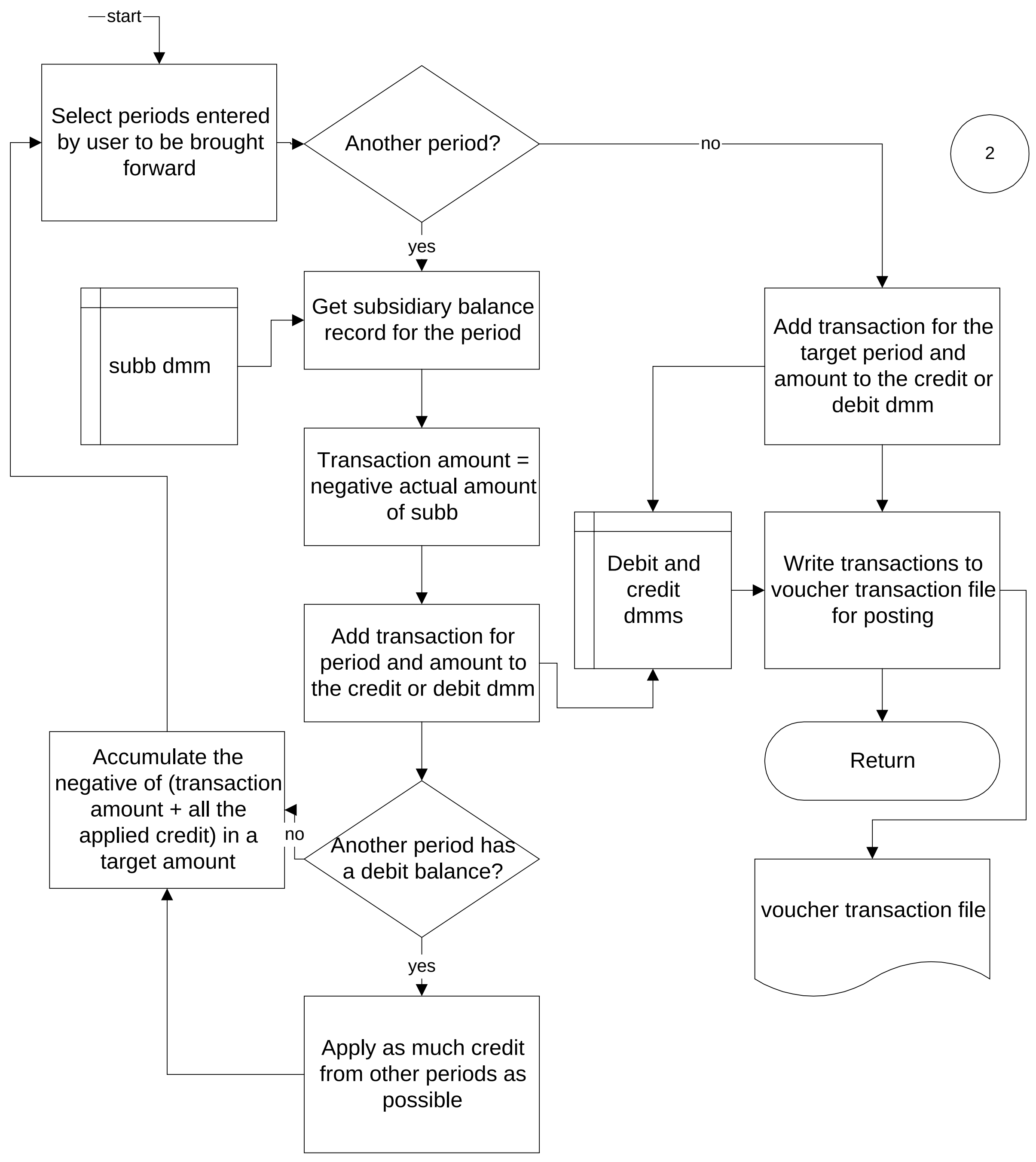
A list of subsidiary accounts to be brought forward

Process Flow

Diagram

The following diagram shows the flow of data in the *sabalfwd* program.


start
Select subsidiary balance records for id and subsidiary
Another subsidiary balance record?
no
yes
Bring debit balances forward?
no
yes
Does user want this period?
no
yes
Bring both debit and credit balances forward
Bring credit balances forward
2
2
Save subsidiary balance record in a list
Stop
subb dmm


start
Select periods entered by user to be brought forward
Another period?
no
yes
2
subb dmm
Get subsidiary balance record for the period
Add transaction for the target period and amount to the credit or debit dmm
Transaction amount = negative actual amount of subb
Debit and credit dmms
Write transactions to voucher transaction file for posting
Add transaction for period and amount to the credit or debit dmm
Return
Accumulate the negative of (transaction amount + all the applied credit) in a target amount
no
Another period has a debit balance?
voucher transaction file
yes
Apply as much credit from other periods as possible

Data Flow Description

The following process describes the data flow in the *sabalfwd* program.

Note: If the user enters specific id numbers, then the program processes each id. If the user elects to process all ids, then the program processes every id for the specified subsidiary. When it concludes processing the ids, the program writes a trailer record to the voucher transaction file.

1. To process an id, the program retrieves subb_recs for the id and subsidiary.
2. If the period on the subb_recs matches a period entered by the user, then the program adds a record to a dmm and retrieves the next subb_rec.
3. How does the user want to process the subsidiary balances?
   - Bring both debit and credit balances forward. The program follows all the steps below.
   - Bring only credit balances forward. The program follows the steps below, except for those referring to debit balances.
4. The program starts at the beginning of the list of *from* periods entered by the user, and performs the following:
   - Retrieves the subb_rec from dmm for the period.
   - Creates a transaction amount equal to the negative actual amount of the subb_rec.
5. If the transaction amount is positive, then the program adds the amount to a debit dmm and accumulates the total debit amount.
6. If the transaction amount is negative, then the program adds the amount to a credit dmm and accumulates the total credit amount.
7. If the program locates another period with a debit bal then it applies as much credit from other bals as possible.
8. The program adds the negative of the transaction amount to all the applied amounts in a target amount, then retrieves the next period.

   Note: If the target amount is less than 0, then the program adds the target amount to the total credit amount and adds a transaction for the target period and the amount to the credit dmm.

   If the target amount is greater than 0, then the program adds the target amount to total debit amount and adds a transaction for the target period and the amount to the debit dmm. The program then writes an entry to the voucher transaction file.

1. Repeat steps 4-8 for each period.

Program Relationships

The *sabalfwd* program creates a file of transactions that becomes input to *filepost*.

Subsidiary Account Balance Forward Parameters

Introduction

CX contains parameters and compilation values for executing the *sabalfwd* program. You can specify parameters to compile *sabalfwd* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *sabalfwd* program.

Parameter Syntax

You can display *sabalfwd* parameters by entering the following: **sabalfwd -,**

The following is the correct usage for running the *sabalfwd* program from the UNIX shell:

sabalfwd -s subsidiary -b balance code -c total code -f list of from sessions -t target session [-i id numbers]

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following lists the parameters for running *sabalfwd*.

-s subsidiary
Required - Specifies a subsidiary for the query (e.g., S/A.)

-b balance code
Required - Specifies a subsidiary balance code (e.g., SB for Session Billing)

-c total code
Required - Specifies a subsidiary total code (e.g., UTUT for undergrad tuition)

-f list of *from* sessions
Required - Specifies the sessions from which you want the program to select subb_recs to roll forward (e.g., SU95 FA95 SP96)

-t target session
Required - Specifies the session to which you want to roll the balances (e.g., SU97)

-i id numbers
Optional - Specifies the id numbers for whom you want to perform the subsidiary balance forwarding

SECTION 18 - SUBSIDIARY ARCHIVE

Overview

Introduction

This section provides reference information about the Subsidiary Archive (*sarc*) program. The General Ledger module uses *sarc* to remove obsolete subsidiary records from the active database to allow CX to operate more efficiently.

Program Features Detailed

This section contains details about the following features of the *sarc* program:

- Process flow
- Parameters
- Table usage

Program Files

All the programs for *sarc* appear in the following directory: $CARSPATH/src/accounting/sarc

Tables Used in the Program

The *sar* program uses the following tables and records:

sar_rec
The Subsidiary Archive records that contain information about the entries, bals and tots that you archive.

suba_rec
The Subsidiary Account records that contain information about the subsidiary number.

subb_rec
The Subsidiary Balance records that contain summary information per period for subsidiary accounts or invoices for accounts payable subsidiary accounts

sube_rec
The Subsidiary Entry records that contain information about postings to the subsidiary accounts

subs_table
The Subsidiary table that defines valid subsidiary codes

subt_rec
The Subsidiary Total records that contain one type of summary information for a subsidiary

subtr_rec
The Subsidiary Transaction records that contain detailed transactions for subsidiary account posting

In addition, sarc creates and calls the following three global data structures as needed:

subb_dmm
A list of subb_recs that are ineligible for archiving

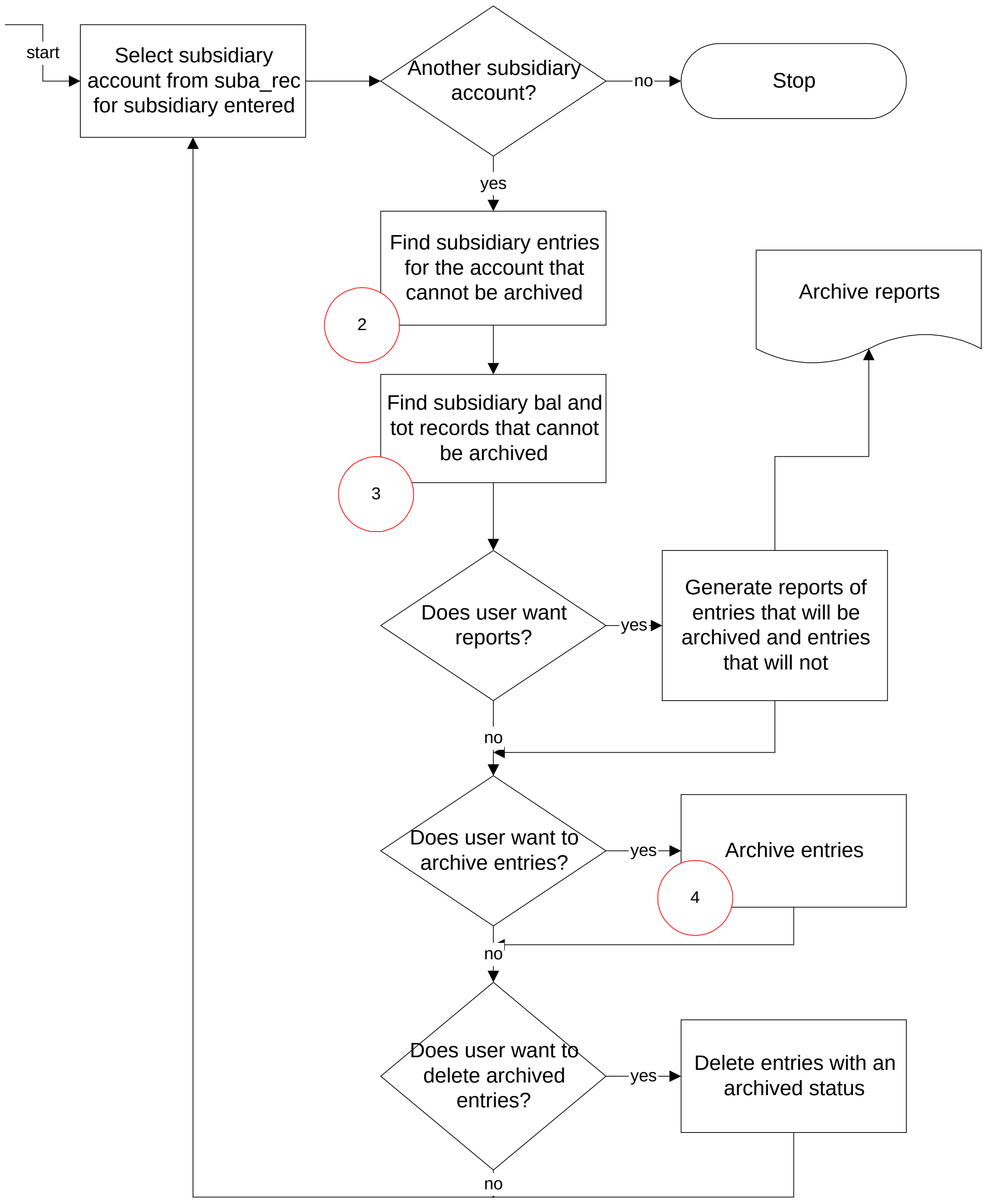
sube_dmm
A list of subsidiary entries for an account

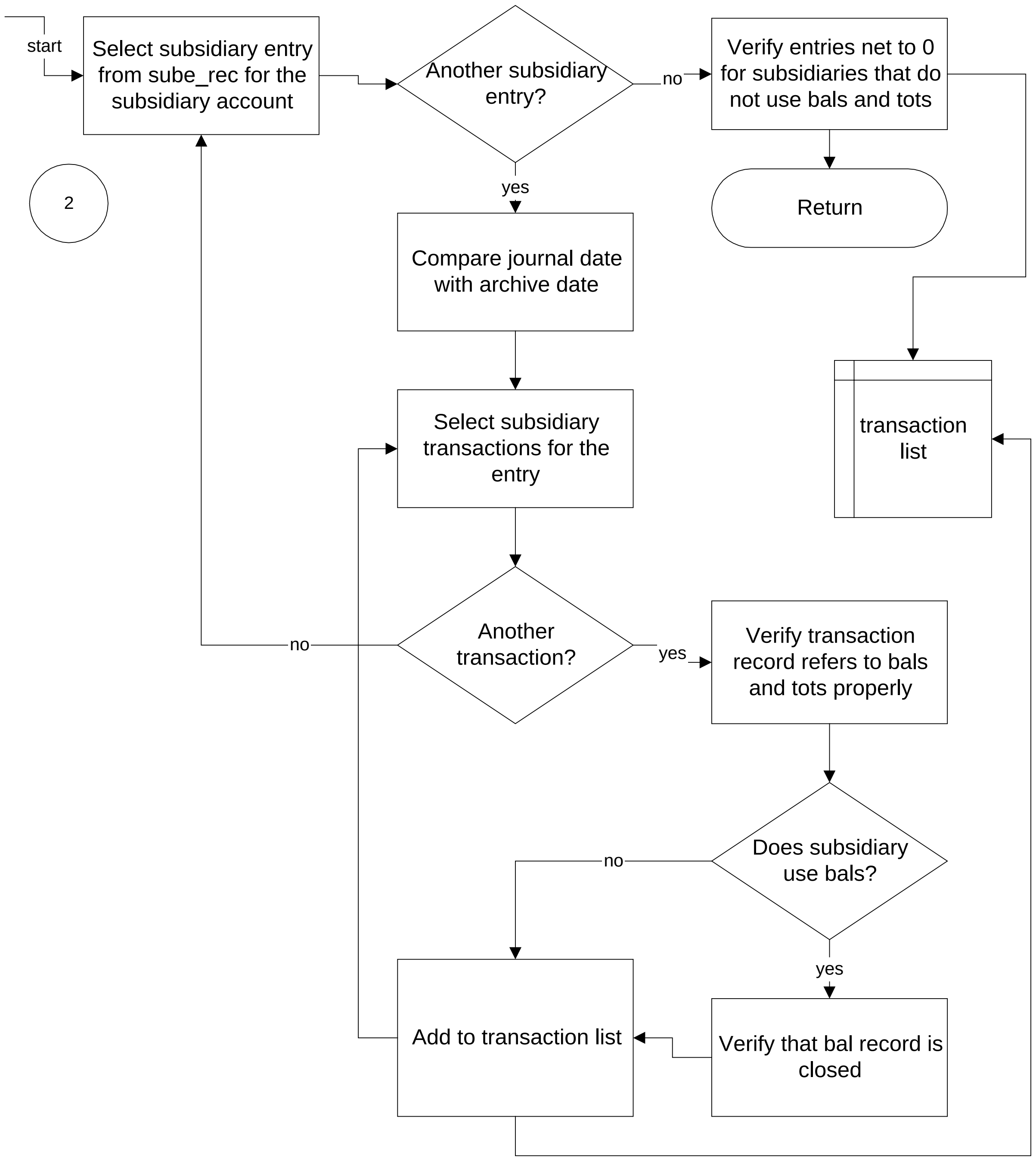
subt_dmm
A list of subt_recs that are ineligible for archiving

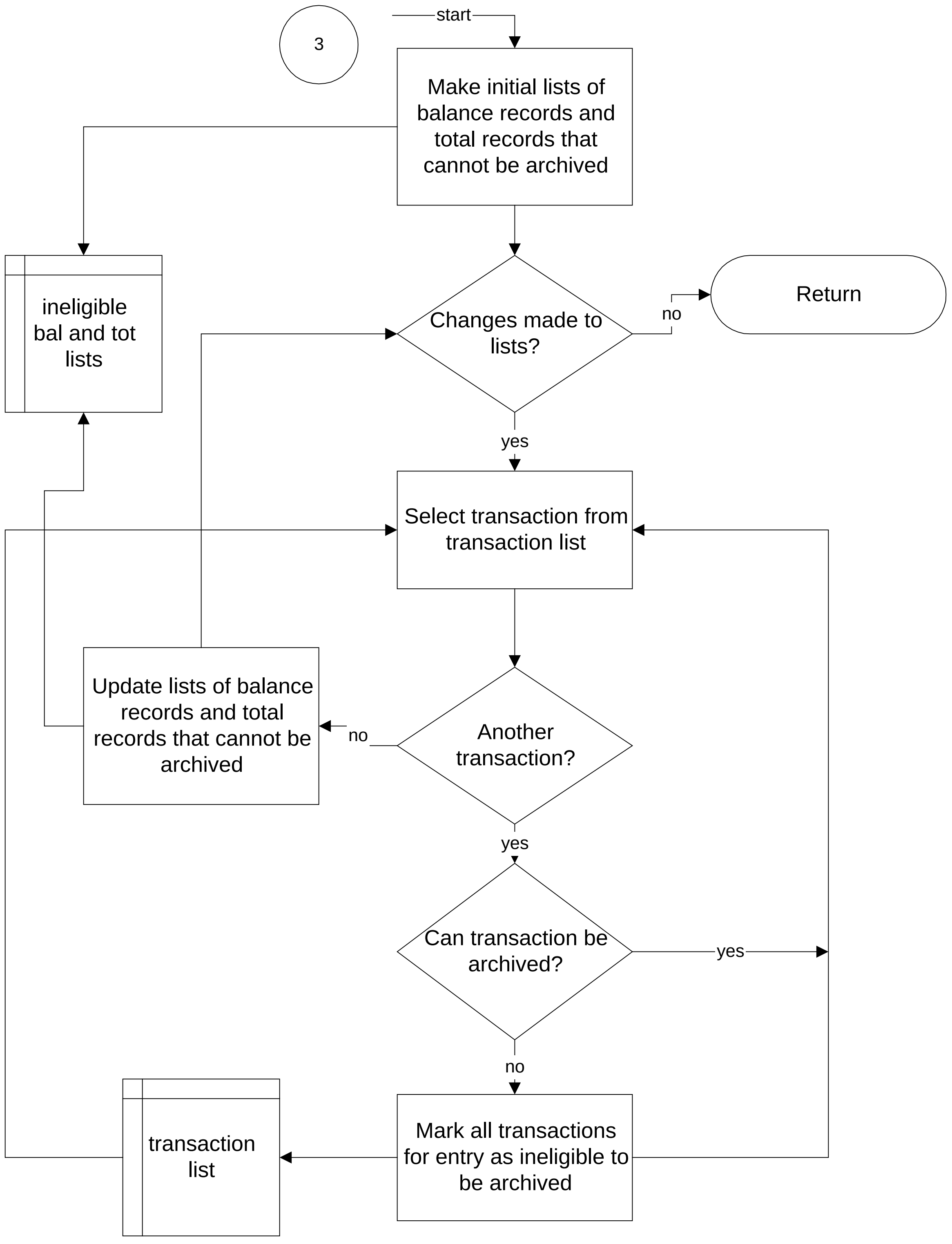
Process Flow

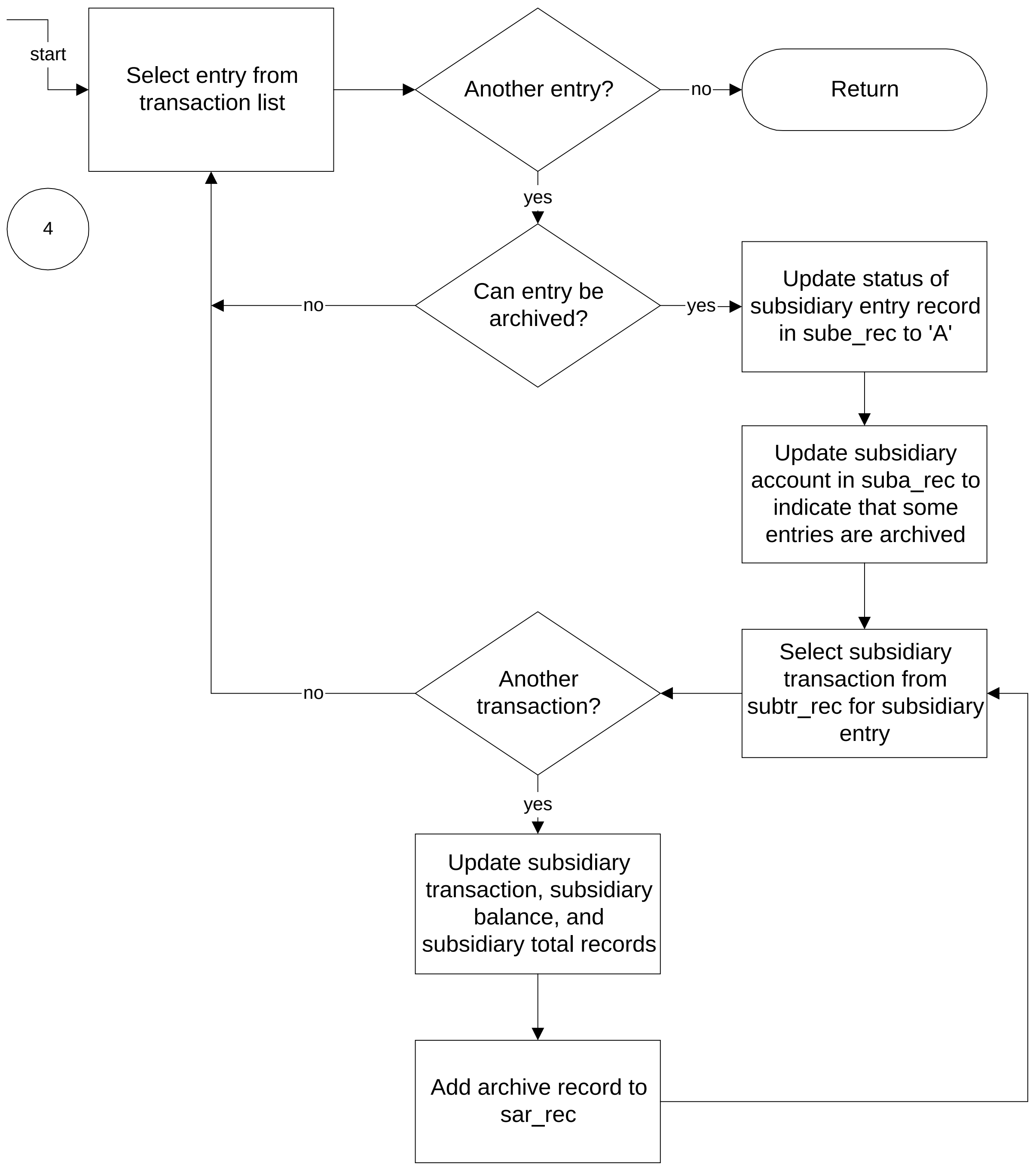
Diagram

The following diagram shows the flow of data in the *sarc* program.


start
Select subsidiary account from suba_rec for subsidiary entered
Another subsidiary account?
no
Stop
yes
Find subsidiary entries for the account that cannot be archived
2
Archive reports
Find subsidiary bal and tot records that cannot be archived
3
Does user want reports?
yes
Generate reports of entries that will be archived and entries that will not
no
Does user want to archive entries?
yes
Archive entries
4
no
Does user want to delete archived entries?
yes
Delete entries with an archived status
no


start
Select subsidiary entry from sube_rec for the subsidiary account
Another subsidiary entry?
no
Verify entries net to 0 for subsidiaries that do not use bals and tots
Return
2
yes
Compare journal date with archive date
Select subsidiary transactions for the entry
transaction list
Another transaction?
no
yes
Verify transaction record refers to bals and tots properly
Does subsidiary use bals?
no
yes
Add to transaction list
Verify that bal record is closed


3
start
Make initial lists of balance records and total records that cannot be archived
ineligible bal and tot lists
Changes made to lists?
no
Return
yes
Select transaction from transaction list
Update lists of balance records and total records that cannot be archived
no
Another transaction?
yes
Can transaction be archived?
yes
no
transaction list
Mark all transactions for entry as ineligible to be archived


start
Select entry from transaction list
Another entry?
no
Return
yes
4
Can entry be archived?
no
yes
Update status of subsidiary entry record in sube_rec to 'A'
Update subsidiary account in suba_rec to indicate that some entries are archived
Select subsidiary transaction from subtr_rec for subsidiary entry
Another transaction?
no
yes
Update subsidiary transaction, subsidiary balance, and subsidiary total records
Add archive record to sar_rec

Data Flow Description

The following process describes the data flow in the *sarc* program.

1. The program selects subsidiary information for the subsidiary entered from subs_table.
2. The program selects the suba_recs for all the subsidiary accounts for the subsidiary entered, or selects the range of accounts entered by the user.
3. When the program locates a subsidiary account, it selects a subsidiary entry from sube_rec of the subsidiary account
4. For each subsidiary entry, the program performs the following processing:
    - Verifies that the journal date is less than or equal to the archive date.
    - Locates the subsidiary transactions for the subsidiary entry.
5. For each subsidiary transaction for the subsidiary entry, the program performs the following processing:
    - If the subsidiary uses bals, verifies if the bal code and bal period are used for the transaction, and checks if the subb_rec for the transaction has been closed. If the subb_record has not been closed, then the program cannot archive the subsidiary entry.
    - If the subsidiary uses tots, verifies if the tot code and tot period are used in the transaction.
    - Adds the transaction to list of entries.
    - Adds to the lists of subb_recs and subt_recs that are ineligible for archiving, using the following logic:
        - If a previous transaction of the same entry is ineligible, then mark the transaction as ineligible.
        - If a transaction is ineligible, then mark all transactions for the entry as ineligible.
    - Builds a list of subb_recs that are ineligible for archiving.
    - Builds a list of subt_recs that are ineligible for archiving.

    Note: If the subsidiary does not use bals and tots, and if the entries do not net to 0.0, then *sarc* does not archive the entries.
6. The program produces reports of entries that can or cannot be archived, based on the requirements of the user.
7. If the user wants to update the status of archived entries to A, the program starts at beginning of the entries list.
8. The program performs the archive on all the entries on the list.
9. If the user wants to delete entries with a status of A, then the program deletes all the appropriate subsidiary entries, transactions, bals and tots.
10. For the archive process for subsidiary entries, the program performs the following:
    - Selects each sube_rec and copies subsidiary entries into the Subsidiary Archive record (sar_rec).
    - Updates the status of the sube_rec to A.
    - Flags the suba_rec as being archived.
11. For the archive process for subsidiary transactions, the program performs the following:
    - Copies the transaction into the sar_rec.
    - Updates the status of the subtr_rec to A.
    - Copies the subb_rec associated with the transaction into the sar_rec.
    - Updates the status of the subb_rec to A.
    - Copies the subt_rec associated with transaction into the sar_rec.

- Updates the status of the subt_rec to A.
- Adds the individual sar_rec to the sar_rec table.

Program Relationships

The *sarc* program does not interact with any other CX programs.

Subsidiary Archive Parameters

Introduction

CX contains parameters and compilation values for executing the sarc program. You can specify parameters to compile *sarc* in a specified manner at the time of execution.

> **Note:** You can also specify compilation values with the includes for the General Ledger module that affect the *sarc* program.

Parameter Syntax

You can display *sarc* parameters by entering the following: **sarc -,**

The following is the correct usage for running the sarc program from the UNIX shell:

```
sarc -s subsidiary -d arc_date [-u] [-b beg_id] [-e end_id] [-o] [-j] [-r] [-v]
```

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following table lists the parameters for running *sarc*.

-s subsidiary
Required - Specifies the subsidiary for which you perform the archive.

-d arc_date
Required - Specifies the archiving date.

-u
Optional - Specifies that you want to set the records’ status to A.

-b beg_id
Specifies the first id number (subsidiary number) that you want to archive.

-e end_id
Optional - Specifies the last id number (subsidiary number) that you want to archive.

-o
Optional - Specifies that you want the program to create an output summary of the archived entries.

-j
Optional - Specifies that you want the program to create an output summary of the entries that are not archived.

-r
Optional - Deletes the records from the database.

-v
Optional - Causes the program to create additional comments and information for debugging purposes.

SECTION 19 - SUBSIDIARY ACCOUNT AUDIT

Overview

Introduction

This section provides reference information about the Subsidiary Account Audit (*saaudit*) program. The General Ledger product uses *saaudit* to reconcile differences between the totals of subsidiary accounts and the related control account. Differences can arise when a General Ledger program ends abnormally, or when system users incorrectly change or delete records using UNIX tools outside the scope of normal CX processing.

Program Features Detailed

This section contains details about the following features of the *saaudit* program:

- Process flow
- Parameters

Program Files

All the program files for *saaudit* appear in the following directory:
$CARSPATH/src/accounting/saaudit

Program Screens

Because *saaudit* is a background process that does not require users to input data, it does not use program screens.

Tables Used in the Program

The *saaudit* program uses the following tables and records:

claim_table
The Claim table that defines account interrelationships for Claim on Cash, Claim on Receivables, and Claim on Payables accounting

ent_table
The Entry table that contains information about valid general ledger entry types

gle_rec
The General Ledger Entry record that contains information about each entry

gltr_rec
The General Ledger Transaction record that contains the amount and account charged for each transaction in an entry

suba_rec
The Subsidiary Account record that contains information about the subsidiary number

subas_table
The Subsidiary Association table that contains information about the relationships between balance and total codes

subb_rec
The Subsidiary Balance records that contain information about a subsidiary balance transaction

sube_rec

The Subsidiary Entry record that contains information about postings to the subsidiary accounts

subs_table
The Subsidiary table that defines valid subsidiary codes

subt_rec
The Subsidiary Total records that contain information about a subsidiary total transaction

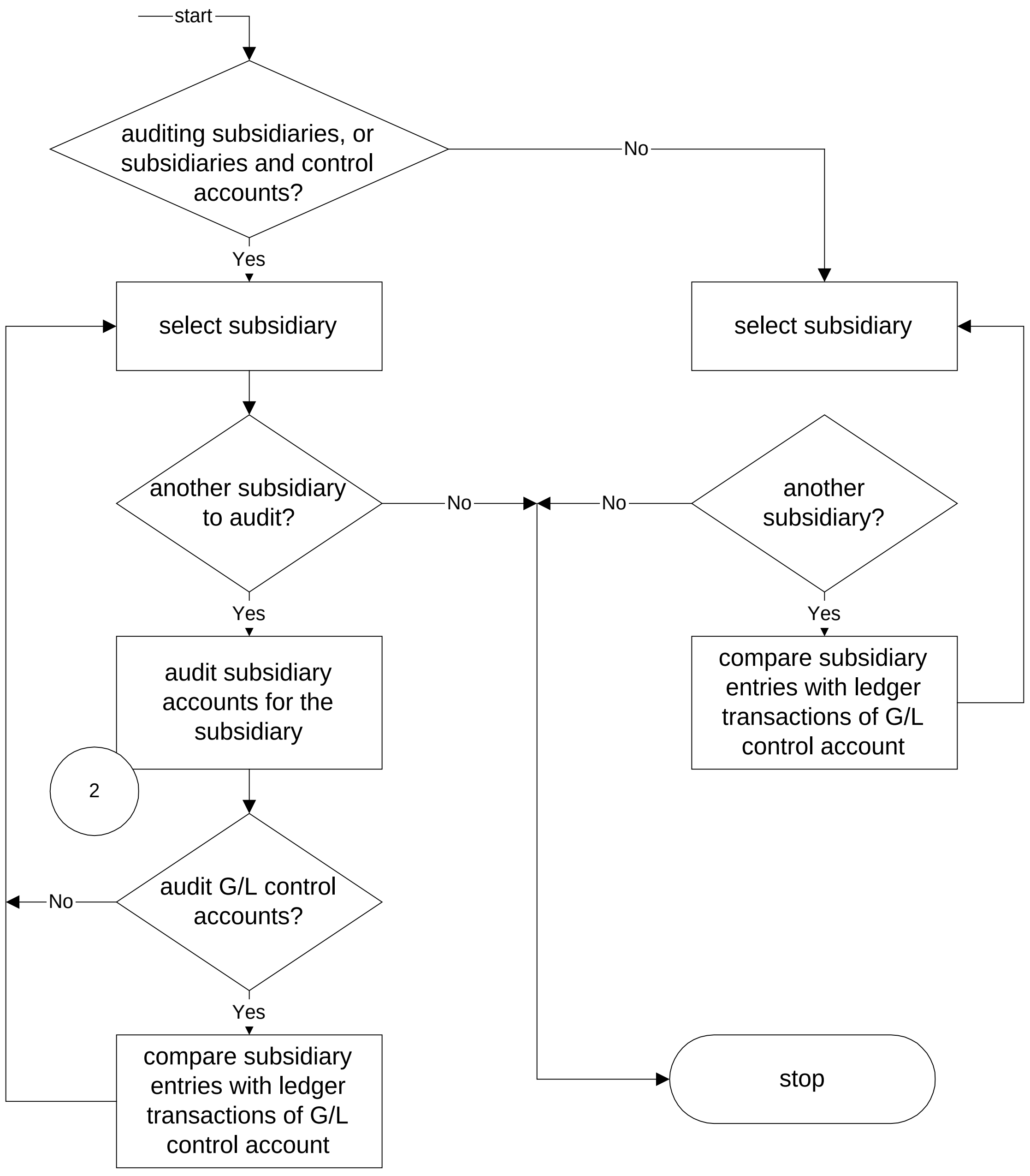
subtr_rec
The Subsidiary Transaction record that contains detailed transactions for subsidiary account posting

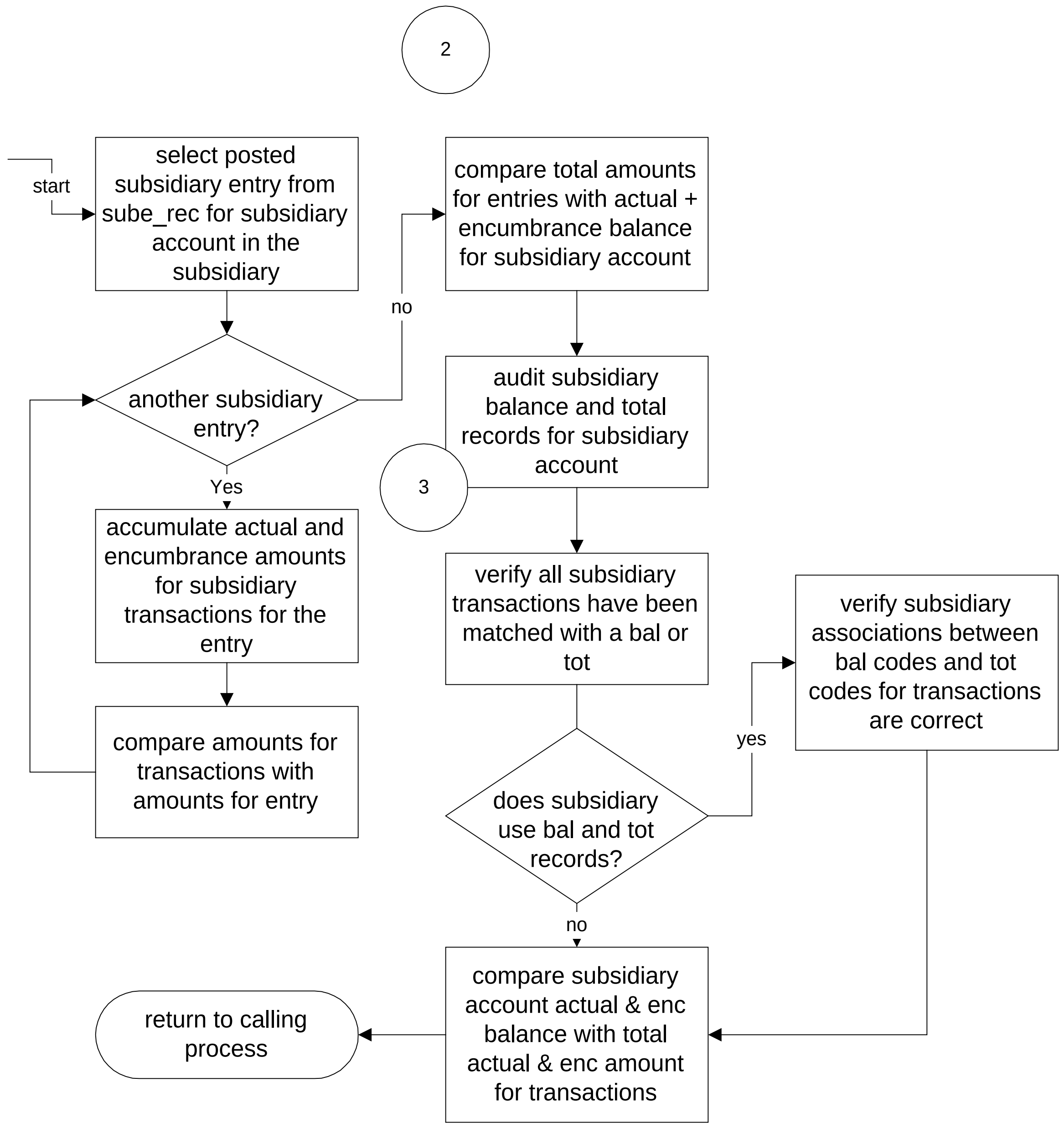
vch_rec
The General Ledger Journal record that contains information about the journal

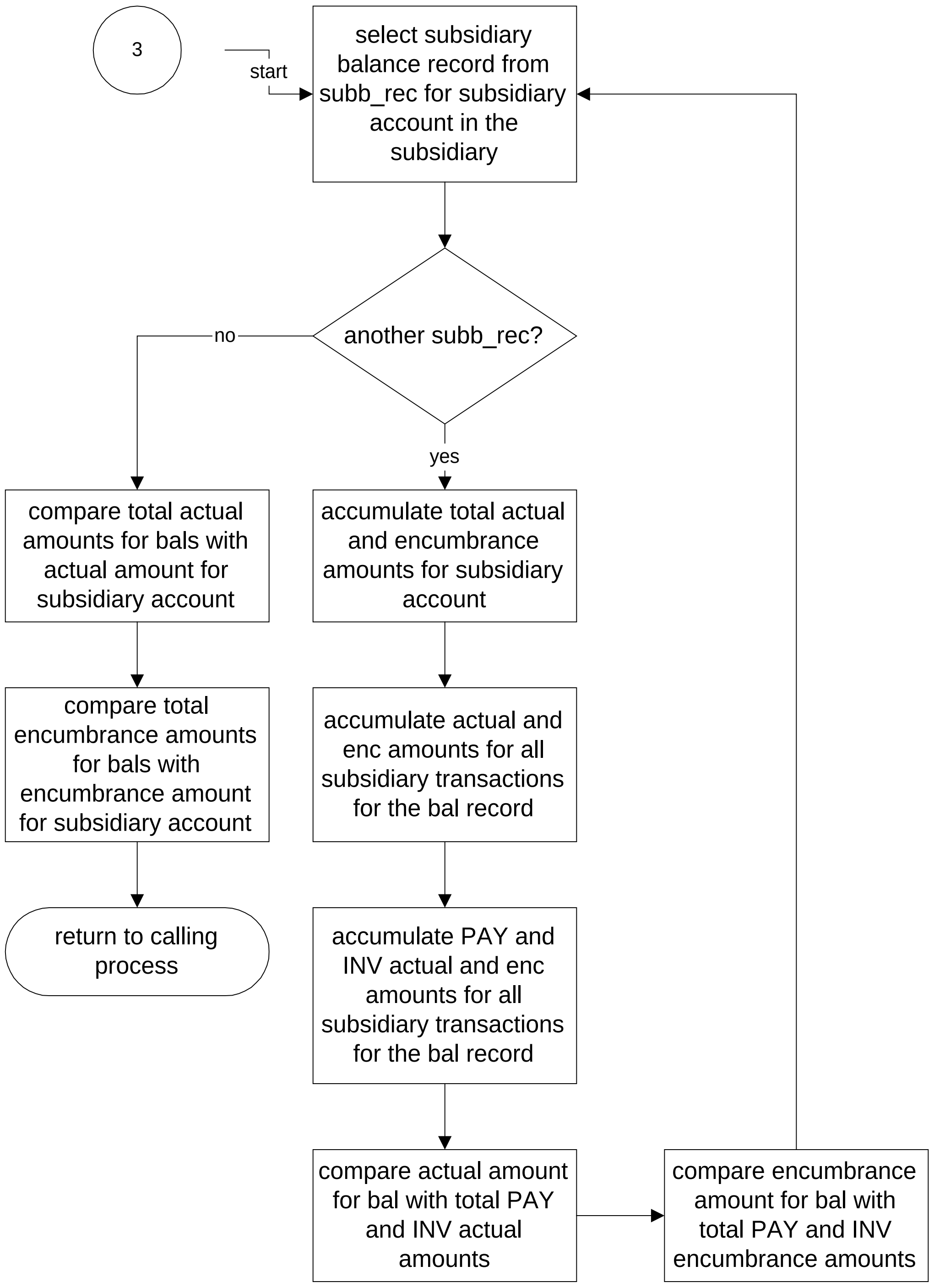
Process Flow

Diagram

The following diagram shows the flow of data in the *saaudit* program.


start
auditing subsidiaries, or subsidiaries and control accounts?
No
Yes
select subsidiary
select subsidiary
another subsidiary to audit?
No
No
another subsidiary?
Yes
Yes
audit subsidiary accounts for the subsidiary
compare subsidiary entries with ledger transactions of G/L control account
2
audit G/L control accounts?
No
Yes
compare subsidiary entries with ledger transactions of G/L control account
stop


2
start
select posted subsidiary entry from sube_rec for subsidiary account in the subsidiary
another subsidiary entry?
Yes
no
accumulate actual and encumbrance amounts for subsidiary transactions for the entry
compare amounts for transactions with amounts for entry
compare total amounts for entries with actual + encumbrance balance for subsidiary account
audit subsidiary balance and total records for subsidiary account
3
verify all subsidiary transactions have been matched with a bal or tot
does subsidiary use bal and tot records?
yes
no
verify subsidiary associations between bal codes and tot codes for transactions are correct
compare subsidiary account actual & enc balance with total actual & enc amount for transactions
return to calling process


3
start
select subsidiary balance record from subb_rec for subsidiary account in the subsidiary
another subb_rec?
no
yes
compare total actual amounts for bals with actual amount for subsidiary account
compare total encumbrance amounts for bals with encumbrance amount for subsidiary account
return to calling process
accumulate total actual and encumbrance amounts for subsidiary account
accumulate actual and enc amounts for all subsidiary transactions for the bal record
accumulate PAY and INV actual and enc amounts for all subsidiary transactions for the bal record
compare actual amount for bal with total PAY and INV actual amounts
compare encumbrance amount for bal with total PAY and INV encumbrance amounts

Data Flow Description

The following describes the data flow in the *saaudit* program.

1. The program loads the subs_table, subas_table and the ent_table into the dmms.
2. Does the user want to audit general ledger control accounts only, subsidiaries only, or subsidiaries *and* general ledger control accounts?
   - If general ledger control accounts only, go to step 5.
   - If subsidiaries only, or subsidiaries *and* general ledger control accounts, go to step 3.
3. To audit subsidiaries, the program selects a subsidiary from the subsidiary dmm and audits the subsidiary accounts.

   Note: If the user wants to audit general ledger control accounts, then the program loads transactions of the general ledger control account for the subsidiary into a dml.
4. The program verifies that the subsidiary entries match the G/L control account entries, then processes the next entry.
5. To audit general ledger control accounts, the program selects each subsidiary from the subsidiary dmm, and performs the following processing:
   - Loads subsidiary entries into a dml
   - Loads ledger transactions of G/L control account for the subsidiary into a dml
   - Verifies that the subsidiary entries match the G/L account entries
6. To audit subsidiary accounts for a subsidiary, the program retrieves the suba_rec for the account, then selects posted subtr_recs for the subsidiary account.
7. The program adds each subtr_rec to a dmm of transactions, and sorts the transaction dmm by entry number.
8. The program selects posted sube_recs for the subsidiary accounts.
9. The program accumulates actual and encumbrance amounts for the subsidiary transactions for each entry.
10. The program compares total amounts for transactions with total amounts for entry, then adds the subsidiary entry into a dml and accumulates the entry amount.
11. The program compares the total amounts for entries with the sum of actual balance + encumbered balance for each account.
12. The program audits subb_recs and subt_recs for subsidiary accounts as follows:
    - Verifies all transactions have been matched with a bal or tot.
    - If the subsidiary uses both bals and tots, then verifies that subsidiary associations between bal codes and tot codes for transactions are correct.
    - Compares the subsidiary account actual balance with the total actual amount for transactions.
    - Compares the subsidiary account encumbered balance with total encumbered amount for transactions.
13. For bals, the program performs the following processing:
    - Selects the subb_rec for the subsidiary account and accumulates actual and encumbrance amounts.
    - Accumulates PAY and INV actual and encumbered amounts for all subsidiary transactions for the bal.
    - Compares bal actual amount with total PAY and INV actual transaction amounts.
    - Compares the bal encumbered amount with total PAY and INV encumbered transaction amounts.

- Compares the total actual amount for bals with the actual amount for the subsidiary account.
- Compares the total encumbered amount for bals with the encumbered amount for the subsidiary account.

14. The program repeats the same logic in step 13 for tot codes, using subt_recs.

Program Relationships

The *saaudit* program does not interact with any other CX programs.

Subsidiary Account Audit Parameters

Introduction

CX contains parameters and compilation values for executing the *saaudit* program. You can specify parameters to compile *saaudit* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger product that affect the *saaudit* program.

Parameter Syntax

You can generate a mail message describing the *saaudit* parameters by entering the following: **saaudit -,**

The following is the correct usage for running the *saaudit* program from the UNIX shell:

```
saaudit -s subs_code -p pass_code [-i id_no] [-l] -b beg_date -e end_date [-m]
```

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following lists the parameters for running *saaudit*.

-s subs_code
Required - Specifies the subsidiary that you want to audit and reconcile (e.g., S/A).

-p pass_code
Required - Designates the type of processing that you want *saaudit* to perform. Valid pass codes are:

- 1 (validate the subsidiary only)
- 2 (validate the control account only)
- 3 (validate both the subsidiary and the control account)

-i id_no
Optional - Designates the number of the subsidiary that you want to reconcile.

-l
Optional - Indicates that you want to perform a limited pass 1 verification (i.e., do not verify bals or tots).

-b beg_date
Required with pass 2 only - Specifies the beginning date for verifying subsidiary entries.

-e end_date
Required with pass 2 only - Specifies the ending date for verifying subsidiary entries.

-m
Optional - Indicates that you want the program to send electronic mail to the user who is running the program.

SECTION 20 - SUBSIDIARY BALANCE STATUS

Overview

Introduction

This section provides reference information about the Subsidiary Balance Status (*subbstat*) program. The General Ledger module uses *subbstat* to set the status flags in the subb_recs. The status flags may require resetting if, for example, your accounting records contain open encumbrances or open invoice reports that show $0.00 items.

Program Features Detailed

This section contains details about the following features of the *subbstat* program:

- Process flow
- Parameters

Program Files

All the program files for *subbstat* appear in the following directory:
$CARSPATH/src/accounting/subbstat

Tables Used in the Program

The *subbstat* program uses the following tables and records:

fscl_cal_rec
The Fiscal Calendar records that define fiscal calendar years and periods

subb_rec
The Subsidiary Balance records that contain information about a subsidiary balance transaction

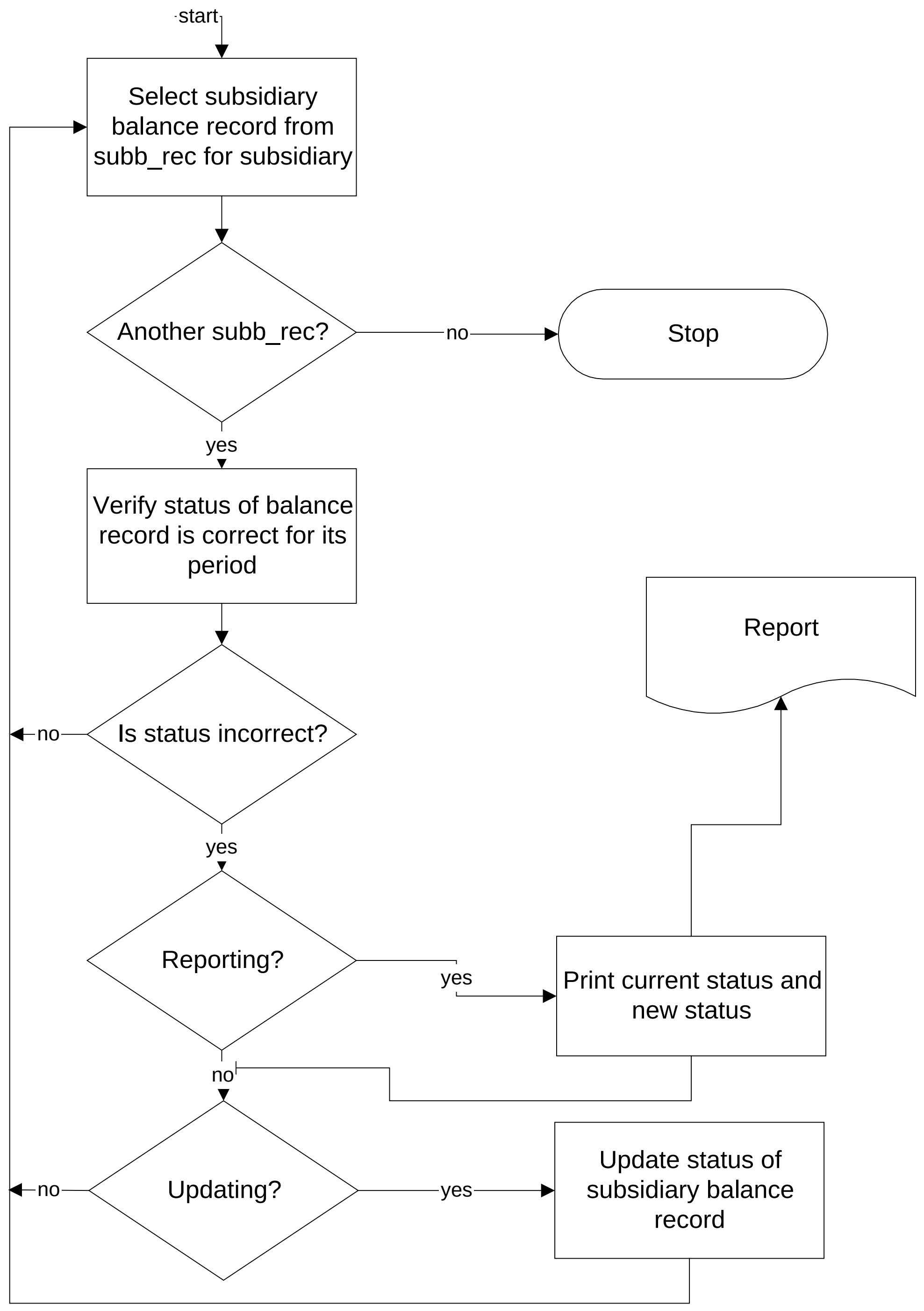
In addition, *subbstat* creates and calls the following global data structure as needed:

fiscal_dml
A structure of fiscal periods for each subsidiary

Process Flow

Diagram

The following diagram shows the flow of data in the *subbstat* program.


start
Select subsidiary balance record from subb_rec for subsidiary
Another subb_rec?
no
Stop
yes
Verify status of balance record is correct for its period
Report
no
Is status incorrect?
yes
Reporting?
yes
Print current status and new status
no
no
Updating?
yes
Update status of subsidiary balance record

Data Flow Description

The following process describes the data flow in the *subbstat* program.

1. The program creates a dml for the Fiscal Calendar records for each subsidiary.

   Note: The top level contains the subsidiary, and the second level contains the dmm for the fiscal calendar periods for the subsidiary.

2. The program selects the subb_recs for the subsidiary entered by the user, locates the period of the subb_rec in the fiscal calendar dml, and uses the following logic to assess the status codes in the subb_recs:
   - If the program cannot locate the period, but the period = INV, then the record must always be closed.
   - If a check has been selected but not paid, then the status of the subb_rec must be L.
   - If the fiscal period always needs to be closed and actual and encumbered amounts = 0.0, then the status of the subsidiary bal must be C.
   - If the fiscal period needs to be closed on the closing date and the actual and encumbered amounts = 0.0 and the run date is after the closing date, then the status of the subsidiary bal must be C.
   - The status of the subsidiary bal should be O for all others.
3. If the current status is not correct and the user wants a report of status changes, then the program prints the previous status and new status.
4. If the user wants to update the status, then the program updates the status of the subb_rec to the new status, and retrieves the next subb_rec.
5. The program repeats steps 2-4 for every subb_rec.

Program Relationships

The *subbstat* program does not interact with any other CX programs.

Subsidiary Balance Status Parameters

Introduction

CX contains parameters and compilation values for executing the *subbstat* program. You can specify parameters to compile *subbstat* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *subbstat* program.

Parameter syntax

The following is the correct usage for running the *subbstat* program from the UNIX shell:

```
subbstat -s subsidiary -d date [-r] [-u]
```

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following table lists the parameters for running *subbstat*.

-s subsidiary
Required - Specifies the subsidiary for which you want to update incorrect statuses.

-d date
Required - Specifies the effective date you want to use to select subb_recs for updating.

-r
Optional - Specifies if you want to produce a report only.

-u
Optional - Specifies if you want to update records. If you do not use this parameter, the program runs in test mode only.

SECTION 21 - VOUCHER

Overview

Introduction

This section provides reference information about the Voucher Processing (*voucher*) program. The General Ledger module uses *voucher* to process journal entries and to create journals. Optionally, end users can use *voucher* to perform queries and cashier functions, although the Accounting Query (*acquery*) and Cashier (*cashier*) programs provide more querying and cash handling functionality.

Program Features Detailed

This section contains details about the following features of the *voucher* program:

- Process flow
- Parameters
- Table relationships
- Program screens

Program Files

All the program files for *voucher* appear in the following directory:
$CARSPATH/src/accounting/voucher

Tables Used in the Program

doc_table
The Document table that contains information about document codes and stations

ent_table
The Entry table that contains information about valid general ledger entry types

gla_rec
The General Ledger Account records that contain the fund, function, object and subfund combinations that your institution has used

gle_rec
The General Ledger Journal Entry records that contain information about each entry

gltr_rec
The General Ledger Transaction records that contain the amount and account charged for each transaction in an entry

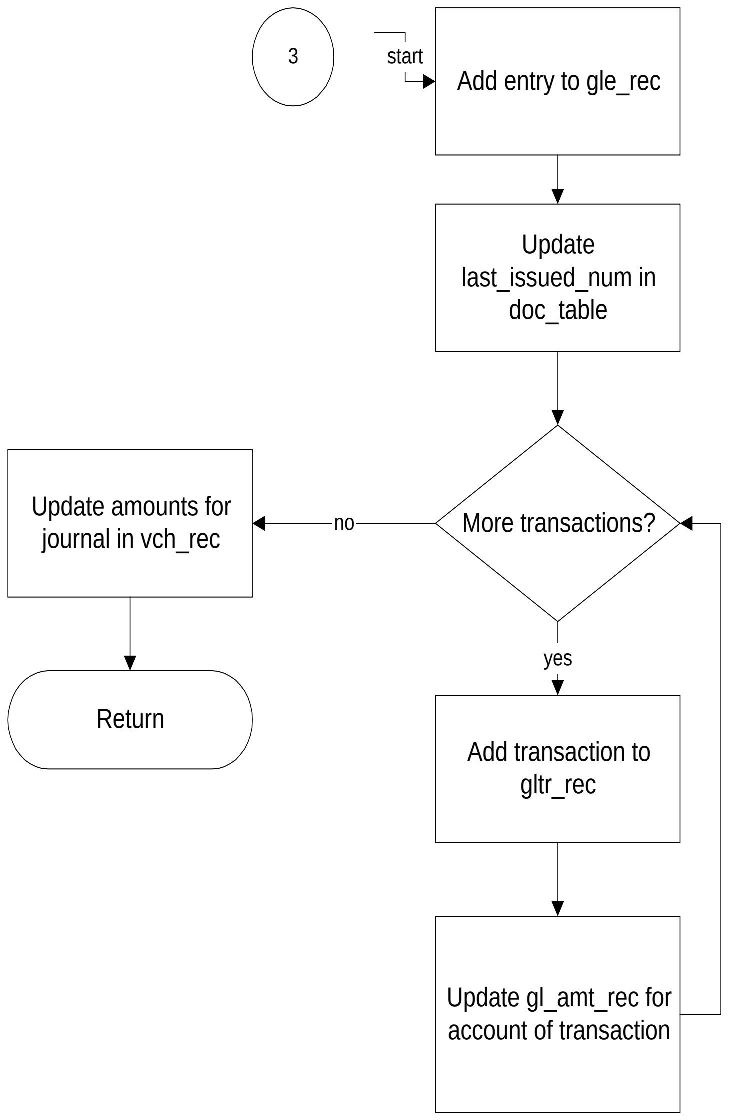
id_rec
The ID record that contains information about each individual or entity in CX

vch_rec
The General Ledger Journal records that contain information about the journal

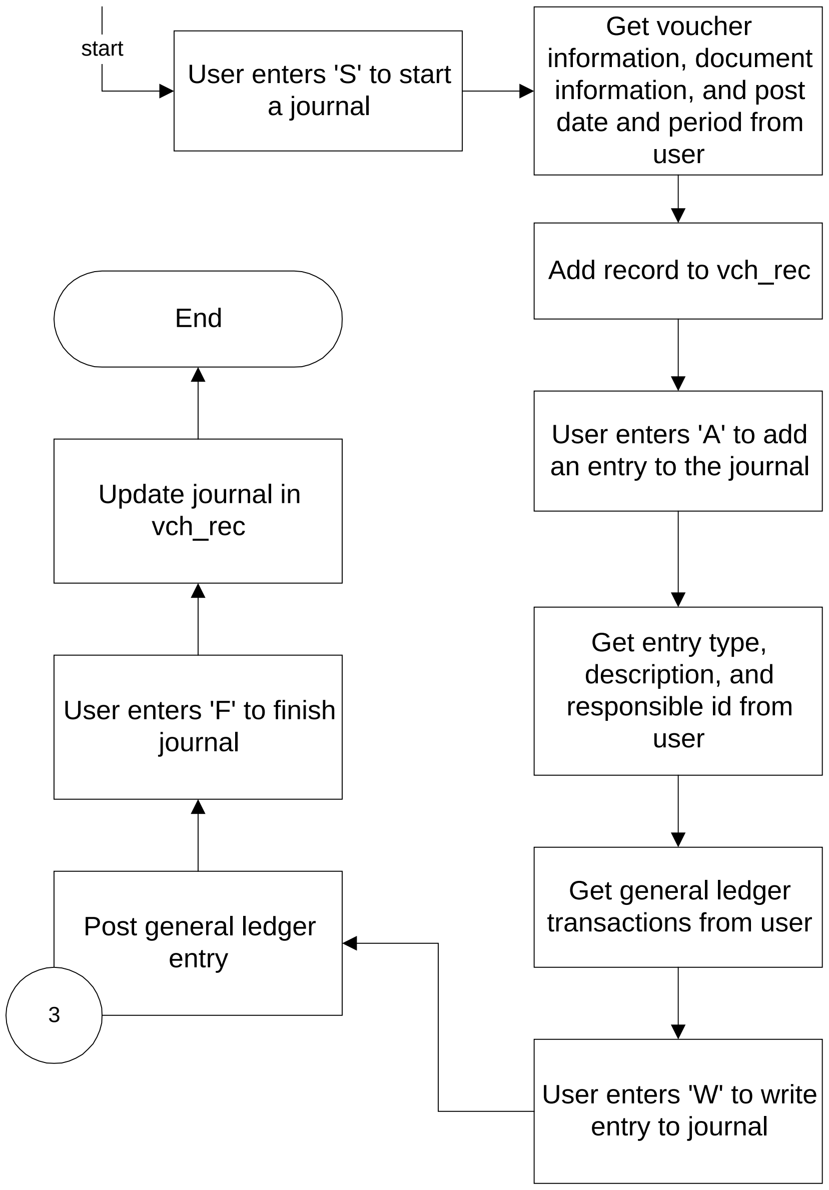
Process Flow

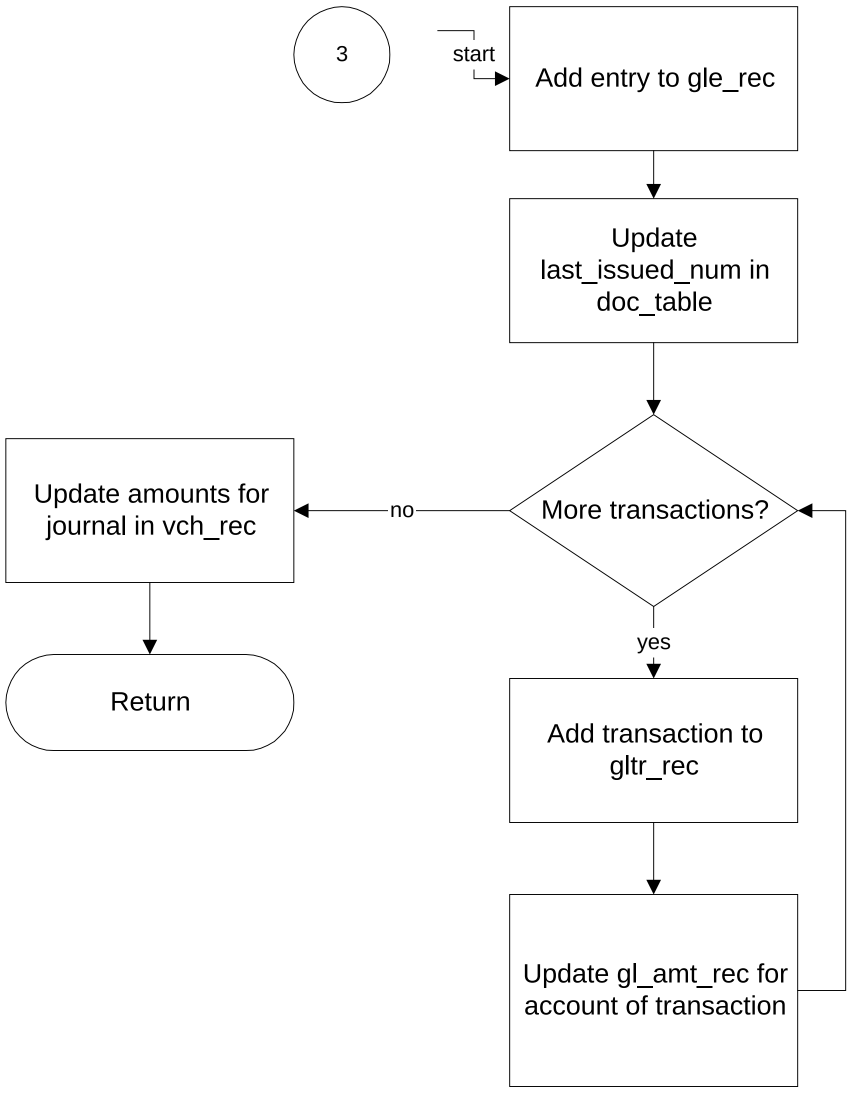
Diagram

The following diagram shows the flow of data in the *voucher* program.



3
start
Add entry to gle_rec
Update last_issued_num in doc_table
More transactions?
no
Update amounts for journal in vch_rec
Return
yes
Add transaction to gltr_rec
Update gl_amt_rec for account of transaction


start
User enters 'S' to start a journal
Get voucher information, document information, and post date and period from user
Add record to vch_rec
User enters 'A' to add an entry to the journal
Get entry type, description, and responsible id from user
Get general ledger transactions from user
User enters 'W' to write entry to journal
Post general ledger entry
3
User enters 'F' to finish journal
Update journal in vch_rec
End


3
start
Add entry to gle_rec
Update last_issued_num in doc_table
More transactions?
no
yes
Update amounts for journal in vch_rec
Return
Add transaction to gltr_rec
Update gl_amt_rec for account of transaction

Data Flow Description

The *voucher* program performs several functions within the General Ledger module, including setting up special purpose accounts, enabling users to create adjusting journal entries, and posting entries to the institution's general ledger. The following processes describe the data flow in each of these functions in the *voucher* program.

Setting up a general ledger account

1. The user selects the G/L command.
2. The program displays G/L Account Entry screen.
3. The program selects the account information for the account entered by the user and validates the account. If the account meets the validation criteria in the tables, the program creates a gla_rec for the account.

Adding an adjusting accounting entry

1. The user selects Start to begin a new journal.
2. The program obtains journal information, document information, and the post date and period from the user.
3. The program adds a vch_rec for the new journal.
4. The user selects Add to add an entry to the journal.
5. The program obtains the entry type, description, and responsible id from the user.
6. The user enters transactions, then selects Write to post the entry to the journal.
7. The program performs the posting.
8. The user selects Finish to finish the journal.
9. The program updates the vch_rec information for the journal.

Posting a general ledger entry

1. The program adds a gle_rec for the entry.
2. The program updates the last_issued_num field in the doc_table.
3. For each general ledger transaction, the program adds a gltr_rec, and updates the gl_amt_rec for the general ledger account in the transaction.
4. The program updates amounts for the journal in its vch_rec.

Program Relationships

The *voucher* program does not interact with any other CX programs.

Voucher Processing Parameters

Introduction

CX contains parameters and compilation values for executing the *voucher* program. You can specify parameters to compile *voucher* in a specified manner at the time of execution.

Note: You can also specify compilation values with the includes for the General Ledger module that affect the *voucher* program.

Parameter syntax

You can display *voucher* parameters by entering the following: **voucher -,**

The following is the correct usage for running the *voucher* program from the UNIX shell:

```
voucher [-a adr] [-d date] [-f vtfile] [-i inst] [-m mode] [-n] [-p printer] [-v jrnl]
```

Parameters that appear in brackets are optional. Parameters that do not appear in brackets are required.

Parameters

The following list contains the parameters for running *voucher*.

-a adr
Optional - Specifies the alternate address run code.

-d date
Optional - Specifies the posting date.

-f vtfile
Currently not in use.

-I inst
Optional - Specifies the instruction for the program. Valid values are as follows:
C (Continue)
S (Start)

-m mode
Optional - Specifies the mode of operation for the journal type. Valid values are as follows:
2 (Interactive)
3 (Student receivable)
4 (Background)
5 (Accounts payable)

-n
Currently not in use.

-p printer
Optional - Specifies the name of the printer that you want to use to print the output from *voucher*.

-v jrnl
Optional - Specifies the journal reference type (e.g., AP or AC), if any.

Program Screens

Introduction

The *voucher* program uses two screens for the features described in this section: one screen for the adding of general ledger accounts, and the other for entering adjusting entries.

Access

The screen files are located in the following directory path:
$CARSPATH/src/accounting/voucher/SCR

Screen Files and Table/Record Usage

The *voucher* screens appear in the following files and use the indicated tables and records:

gladd
Contains the screen to use to add special purpose general ledger accounts.
Tables/Records: gla_rec, subs_table

voucher
Contains the Journal Entry screen.
Tables/Records: doc_table, ent_table, gla_rec, gle_rec, gltr_rec, id_rec, pay_frm_table, suba_rec, vch_rec, vch_table

SECTION 22 - VOUCHER RECOVERY

Overview

Introduction

This section provides reference information about the Voucher Recovery (*vchrecover*) program. The General Ledger module uses *vchrecover* to recover journals that have been affected by system failure.

Program Features Detailed

This section contains details about the following features of the *vchrecover* program:

- Process flow
- Parameters
- Table relationships

Program Files

All the program files for vchrecover appear in the following directory:
$CARSPATH/src/accounting/vchrecover

Tables Used in the Program

The *vchrecover* program uses the following tables and records

chrecon_rec
The Cashier Reconciliation records that contain information about the reconciliation status of General Ledger transactions

doc_table
The Document table that contains information about document codes and stations

gle_rec
The General Ledger Journal Entry records that contain information about each entry

gltr_rec
The General Ledger Transaction records that contain the amount and account charged for each transaction in an entry

sube_rec
The Subsidiary Entry records that contain information about postings to the subsidiary accounts

subtr_rec
The Subsidiary Transaction records that contain detailed transactions for subsidiary account posting

vch_rec
The General Ledger Journal records that contain information about the journal

In addition, *vchrecover* creates and calls the following four global data structures as needed:

gltr_dmm
A list of G/L transactions for last G/L entry of voucher being recovered

subs_dml
The structure of subsidiary entries and transactions

vch_err_dmm

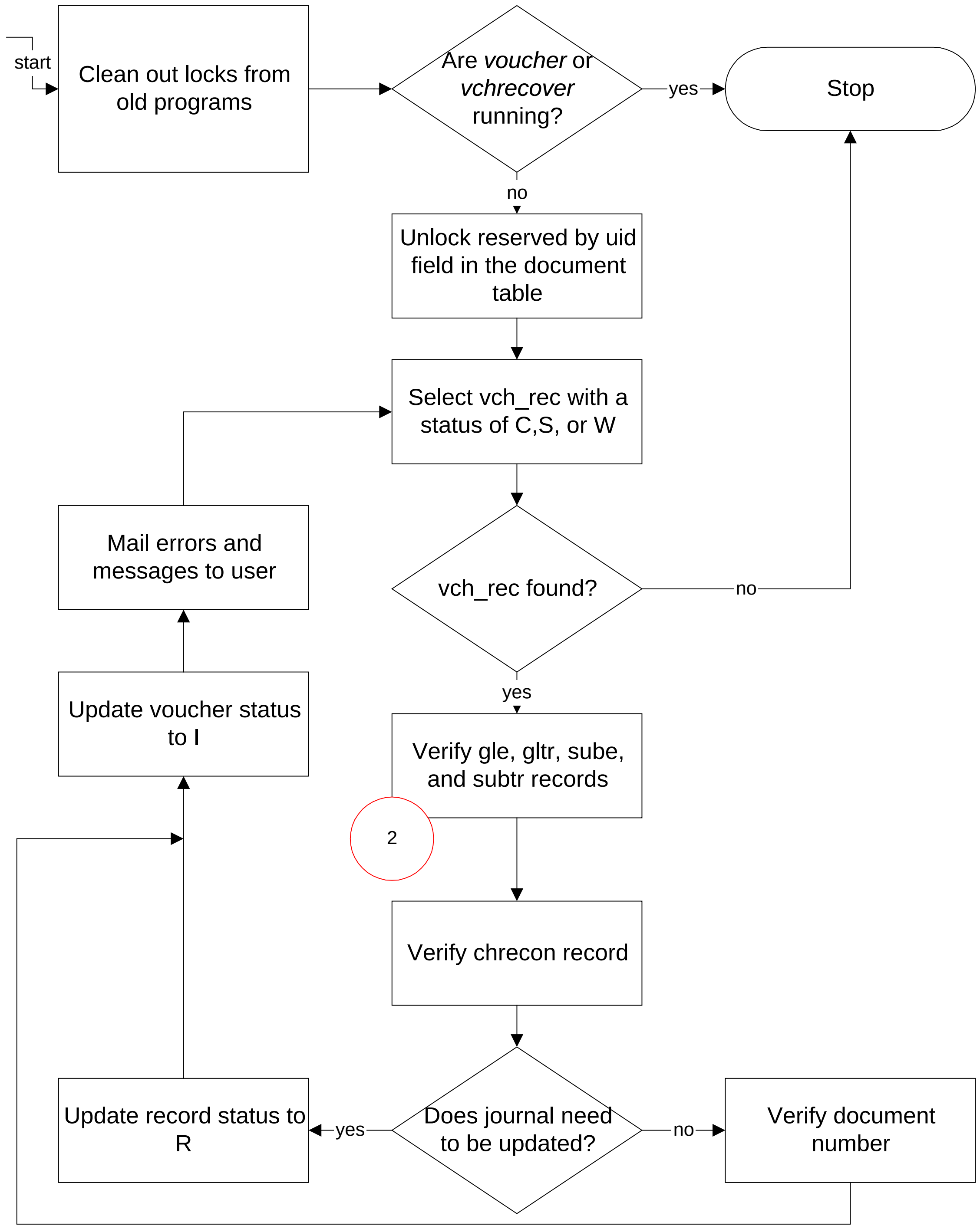
A list of errors with journals to be mailed to user

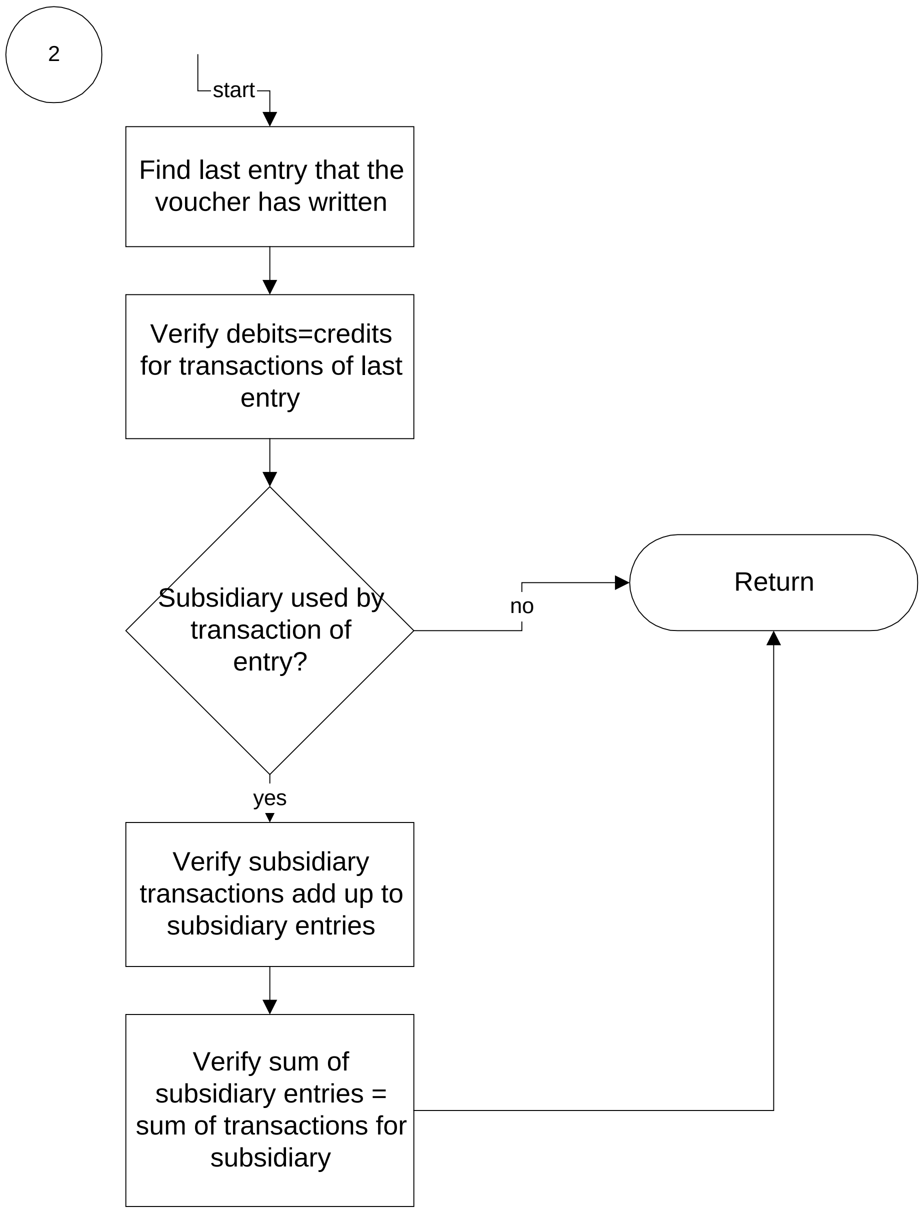
prog_err_dmm
A list of program errors to be mailed to user

Process Flow

Diagram

The following diagram shows the flow of data in the *vchrecover* program.


start
Clean out locks from old programs
Are voucher or vchrecover running?
yes
Stop
no
Unlock reserved by uid field in the document table
Select vch_rec with a status of C,S, or W
Mail errors and messages to user
vch_rec found?
no
yes
Update voucher status to I
Verify gle, gltr, sube, and subtr records
2
Verify chrecon record
Update record status to R
yes
Does journal need to be updated?
no
Verify document number


2
start
Find last entry that the voucher has written
Verify debits=credits for transactions of last entry
Subsidiary used by transaction of entry?
no
Return
yes
Verify subsidiary transactions add up to subsidiary entries
Verify sum of subsidiary entries = sum of transactions for subsidiary

Data Flow Description

The following process describes the data flow in the *vchrecover* program.

1. The program ensures that any locks from old programs have been removed from the lock file.
2. The program checks if either *vchrecover* or *voucher* is already running. If either program is running, *vchrecover* does not run.
3. The program unlocks the reserved by uid field in the doc_table and retrieves vch_recs with C,S,or W as the status.
4. For every vch_rec found, the program performs the following verifications:
    - Accuracy of gle, gltr, sube, and subtr records.
    - Existence of the chrecon_rec, if needed.
5. If the journal needs to be updated, then the program updates records to R status, and checks the document number.
6. If the journal does not require updating, the program checks the document number.
7. The program updates the journal to Incomplete status.

    Note: Regardless of whether the journal requires updating, the program changes all journals with a status of C, S or W so they have a status of I (Incomplete).
8. The program mails errors and messages to the user.

To verify the gle, gltr, sube, and subtr records, *vchrecover* performs the following:

1. Retrieves the last gle_rec according to the vch_rec.
2. Attempts to find the next gle_rec (i.e., a gle_rec that is not reflected in vch_rec). If the program cannot locate a next record, then the entry located above is the last entry that the journal has written.
3. Retrieves all gltr_recs for the last general ledger entry.
4. If the transaction uses a subsidiary control account, then the program accumulates the amount for the subsidiary, and checks if total debits = total credits.
5. If the general ledger entry uses a subsidiary, then the program retrieves all subsidiary entries and subsidiary transactions for the general ledger entry.
6. For each subsidiary entry, the program checks if the total for the transactions adds up to the subsidiary entry amount.
7. The program accumulates the amount for the subsidiary.

    Note: If the program cannot locate any subsidiary entries, then the journal is not completely posted.
8. The program checks if the sum of the subsidiary entries equals the sum of the general ledger transactions for each subsidiary account referenced.

    Note: If the program cannot locate any gltr_recs, then journal is not completely posted.

Program Relationships

The *vchrecover* program does not interact with any other CX programs.

Voucher Recovery Parameters

The *vchrecover* program does not require any processing parameters.

SECTION 23 - MENUS, SCREENS AND SCRIPTS

Overview

Introduction

This section provides reference information on the following features of the General Ledger module:

- Menu source files
- Menu option files
- PERFORM screens
- SQL scripts
- Csh scripts

Directory Locations

The features detailed in this section are located in the following directory paths:

- *Menu source files*:
  - $CARSPATH/menusrc/fiscal/finacctg/finrptg
  - $CARSPATH/menusrc/fiscal/finacctg/fintables
  - $CARSPATH/menusrc/fiscal/finacctg/glmaint
  - $CARSPATH/menusrc/fiscal/finacctg/jrnlproc
  - $CARSPATH/menusrc/fiscal/finacctg/prdproc
- *Menu option files*:
  - $CARSPATH/menuopt/accounting/informers
  - $CARSPATH/menuopt/accounting/others
  - $CARSPATH/menuopt/accounting/programs
  - $CARSPATH/menuopt/accounting/reports
  - $CARSPATH/menuopt/accounting/screens
  - $CARSPATH/menuopt/accounting/scripts
- *PERFORM screens*:
  - $CARSPATH/modules/accounting/screens
- *SQL scripts*:
  - $CARSPATH/modules/accounting/informers
- *Csh scripts*:
  - $CARSPATH/modules/accounting/scripts

General Ledger Menus

Introduction

The CX menu source (menusrc) directory path contains definitions of the CX menu structure. Specifically, the $CARSPATH/menusrc/fiscal directory path contains definitions for General Ledger menus. The following directories corresponding to General Ledger appear in this path:

- $CARSPATH/menusrc/fiscal/finacctg
- $CARSPATH/menusrc/fiscal/finacctg/finrptg
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/acctrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/assocrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/cntrcombrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/cntrrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/cusfstmt
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/finstmt
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/grntrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/miscrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/projrpts
- $CARSPATH/menusrc/fiscal/finacctg/finrptg/subsrpts
- $CARSPATH/menusrc/fiscal/finacctg/fintables
- $CARSPATH/menusrc/fiscal/finacctg/glmaint
- $CARSPATH/menusrc/fiscal/finacctg/glmaint/audits
- $CARSPATH/menusrc/fiscal/finacctg/jrnlproc
- $CARSPATH/menusrc/fiscal/finacctg/jrnlproc/asciipost
- $CARSPATH/menusrc/fiscal/finacctg/jrnlproc/jrnlmaint
- $CARSPATH/menusrc/fiscal/finacctg/jrnlproc/recurring
- $CARSPATH/menusrc/fiscal/finacctg/jrnlproc/sae
- $CARSPATH/menusrc/fiscal/finacctg/prdproc
- $CARSPATH/menusrc/fiscal/finacctg/prdproc/990rpts
- $CARSPATH/menusrc/fiscal/finacctg/prdproc/glclsg
- $CARSPATH/menusrc/fiscal/finacctg/prdproc/subarch
- $CARSPATH/menusrc/fiscal/finacctg/prdproc/subbalfwd

Each directory above contains a *menudesc* file, specifying what menu options appear in a menu. Specific menu options, however, are defined in the menu option (menuopt) directory path.

Menu Options

The following table associates each General Ledger program, screen, and script menu option and corresponding menuopt file and identifies the menuopt locations and what the menu option accesses.

Note: The menu options appear in the table in the order of their appearance in the above menusrc directories.

Menu	Menu option	Menuopt file	Accesses ...
Fiscal Management: Accounting Main Menu	Accounting Query	$CARSPATH/menuopt/ accounting/programs/ acqu.p	Program: *acquery* Parameters passed: -s (start on subsidiary side using specified subsidiary) -r (subsidiary restrictions) -l (lock into one side)

Menu	Menu option	Menuopt file	Accesses ...
			-p (printer for statement output)
	Budget Review	$CARSPATH/menuopt/ accounting/programs/ bgtr	Program: *bgtreview* Parameters passed: -y (fiscal year) -m (fiscal period) -f (display frequency of account balances) -p (output printer)
Accounting: Reports menu	Initialize Report Indexes	$CARSPATH/menuopt/ accounting/informers/ addglatemp	SQL script: addglatemp Parameters passed: DFISCAL_YR (fiscal year)
	Review Report Indexes	$CARSPATH/menuopt/ accounting/screens/ glatemp	PERFORM screen: FRMPATH/accounting/ glatemp
Accounting: Objects menu	Detail	$CARSPATH/menuopt/ accounting/others/ acctdtl	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling)
	Detail by Funds	$CARSPATH/menuopt/ accounting/others/ acctdtlfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN

Menu	Menu option	Menuopt file	Accesses ...
			(subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling)
	Detail by Month	$CARSPATH/menuopt/accounting/others/acctdtlmon	<u>Report</u> <u>Parameters passed</u>: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (produce summary report)
	Summary	$CARSPATH/menuopt/accounting/others/acctsum	<u>Report</u> <u>Parameters passed</u>: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule(PP_SUMMARY (summary

Menu	Menu option	Menuopt file	Accesses ...
			report)
	Summary by Funds	$CARSPATH/menuopt/ accounting/others/ acctsumfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)
	Transactions		Report Parameters passed: PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRAN (detail/summary transactions) PP_SUBT_PRGLS (function/subfund subtotaling)
Accounting:	Balance Sheet	$CARSPATH/menuopt/	Report

Menu	Menu option	Menuopt file	Accesses ...
Associated Objects Reports menu		accounting/others/ balshtasc	Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_ASSOC_OBJ (association code) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_NONDSPL (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)
	Balance Sheet by Funds	$CARSPATH/menuopt/ accounting/others/ balshtasf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_ASSOC_OBJ (association code) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_NONDSPL (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)
	Revenue/ Expense Detail	$CARSPATH/menuopt/ accounting/others/ revexpdasc	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_ASSOC_OBJ (association code) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_NONDSPL (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Rev/Exp Detail by Funds	$CARSPATH/menuopt/ accounting/others/ revexpdasf	Report Parameters passed: PP_COL_OUTPUT_TYPE

Menu	Menu option	Menuopt file	Accesses ...
			(formtype) PP_ASSOC_OBJ (association code) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_NONDSPL (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Revenue/ Expense Summary	$CARSPATH/menuopt/ accounting/others/ revexpasc	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_ASSOC_OBJ (association code) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_NONDSPL (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Rev/Exp Summary by Funds	$CARSPATH/menuopt/ accounting/others/ revexpasf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_ASSOC_OBJ (association code) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_NONDSPL (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Object Transactions	$CARSPATH/menuopt/ accounting/others/ accttrasc	Report Parameters passed: PP_ASSOC_OBJ (associated object code) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_NONDSPL_OBJ

Menu	Menu option	Menuopt file	Accesses ...
			(nondisplay of selected objects) PP_TRAN (detail/summary transactions) PP_SUBT_PRGLS (function/subfund subtotaling)
	Function Transactions	$CARSPATH/menuopt/ accounting/others/ cntrtrasc	Report Parameters passed: PP_ASSOC_OBJ (associated object code) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRAN (detail/summary transactions)
Accounting: Combined Function Reports menu	Object Detail	$CARSPATH/menuopt/ accounting/others/ acctdtlcom	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FUNC_COMB_RAN (combined function code range) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Object Detail by Funds	$CARSPATH/menuopt/ accounting/others/ acctdtlcmf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag)

Menu	Menu option	Menuopt file	Accesses ...
			PP_FUNC_COMB_RAN (combined function code range) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Function Detail	$CARSPATH/menuopt/ accounting/scripts/ cntrdtlcom	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FUNC_COMB_RAN (combined function code range) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group) PP_SUMMARY (summary report)
	Function Detail by Funds	$CARSPATH/menuopt/ accounting/others/ cntrdtlcmf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible

Menu	Menu option	Menuopt file	Accesses ...
			person flag) PP_FUNC_COMB_RAN (combined function code range) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Function Summary	$CARSPATH/menuopt/ accounting/others/ cntrsumcom	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FUNC_COMB_RAN (combined function code range) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Function Summary by Funds	$CARSPATH/menuopt/ accounting/others/ cntrsumcmf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible

Menu	Menu option	Menuopt file	Accesses ...
			person flag) PP_FUNC_COMB_RAN (combined function code range) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
Accounting: Function Reports menu	Detail	$CARSPATH/menuopt/ accounting/others/ cntrdtl	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group)
	Detail by Funds	$CARSPATH/menuopt/ accounting/others/ cntrdtlfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type)

Menu	Menu option	Menuopt file	Accesses ...
			PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group)
	Detail by Month	$CARSPATH/menuopt/ accounting/others/ cntrdtlmon	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Summary	$CARSPATH/menuopt/ accounting/others/ cntrsum	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ

Menu	Menu option	Menuopt file	Accesses ...
			(nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group) PP_SUMMARY (summary report)
	Summary by Funds	$CARSPATH/menuopt/accounting/others/cntrsumfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group) PP_SUMMARY (summary report)
	Transactions	$CARSPATH/menuopt/accounting/others/cntrtrans	Report Parameters passed: PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRAN (detail or

Menu	Menu option	Menuopt file	Accesses ...
			summary transactions)
	Detail Transactions	$CARSPATH/menuopt/ accounting/scripts/ sortcntr	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRANS (detail or summary transactions) PP_SUBT_SCHGRP (subtotaling by schedule or group)
	Profit Center	$CARSPATH/menuopt/ accounting/others/ cntrprf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRANS (detail or summary transactions) PP_SUBT_BLGRP (subtotaling by block or group)

Menu	Menu option	Menuopt file	Accesses ...
			PP_PERSON_PRNT (print responsible person's name)
Accounting: Custom Financial Statements menu	Fin Stmnt Struct Entry	$CARSPATH/menuopt/ accounting/programs/ fgen	Program: *fingen* Parameters passed: -y (fiscal year) -p (printer)
	Struct Fin Stmt Generation	$CARSPATH/menuopt/ accounting/programs/ frpt	Program: *finrpt* Parameters passed: -r (report code) -y (fiscal year) -Y (comparative fiscal year) -m (starting period) -M (ending period) -A (actual amount flag) -E (encumbrance amount flag) -B (budget amount flag) -x (print G/L account exception list flag)
	Fin Set Table	$CARSPATH/menuopt/ accounting/screens/ tfinset	PERFORM screen
	Fin Rpt Format Entry	$CARSPATH/menuopt/ accounting/programs/ finfmt	Program: finfmt Parameters passed: -p (printer)
Accounting: Financial Statement Reports menu	Balance Sheet	$CARSPATH/menuopt/ accounting/others/ acctsum.bs	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRANS (detail or summary transactions) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule)

Menu	Menu option	Menuopt file	Accesses ...
			PP_SUMMARY (summary report)
	Balance Sheet by Funds	$CARSPATH/menuopt/ accounting/others/ acctsum.bf	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person flag) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)
	Revenue/ Expense Detail	$CARSPATH/menuopt/ accounting/others/ revexpd.rx	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)

Menu	Menu option	Menuopt file	Accesses ...
	Revenue/ Expense Detail by Funds	$CARSPATH/menuopt/ accounting/others/ revexpdfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Revenue/ Expense Summary	$CARSPATH/menuopt/ accounting/others/ revexp.rx	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Rev/ Exp Summary by Funds	$CARSPATH/menuopt/ accounting/others/ revexpfd.x	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code

Menu	Menu option	Menuopt file	Accesses ...
			range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Trial Balance	$CARSPATH/menuopt/accounting/others/trialbal	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Trial Balance by Funds	$CARSPATH/menuopt/accounting/others/trialbal.f	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary

Menu	Menu option	Menuopt file	Accesses ...
			report)
	G/L Net Asset Excptn Rpt	$CARSPATH/menuopt/ accounting/reports/ exntast	Report Parameters passed: -f (formtype)
Accounting: Grants menu	Grant Detail Report	$CARSPATH/menuopt/ accounting/others/ grntdtlr	Report Parameters passed: PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_BEG (beginning fiscal period) PP_FS_YR_BEG (beginning fiscal year) PP_FS_END (ending fiscal period) PP_FS_YR_END (ending fiscal year) PP_FUND (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT-SCH (subtotaling by schedule) PP_SUMMARY (summary report)
Accounting: Miscellaneous Reports menu	Cash Flow History	$CARSPATH/menuopt/ accounting/others/ cashflow	Report Parameters passed: PP_FS_RAN (fiscal year range) PP_FS_YR (fiscal year) PP_FUND (fund code range) PP_TOTAL (totals only for summary reports)
	Cash/Main Funds	$CARSPATH/menuopt/ accounting/others/ cashrpt	Report Parameters passed: PP_COL_OUTPUT (formtype) PP_FS_CODE (fiscal period code) PP_FS_YR (fiscal year)
	Cash/Specific Funds	$CARSPATH/menuopt/ accounting/others/ cashrpt.US	PP_COL_OUTPUT (formtype) PP_FS_CODE (fiscal period code) PP_FS_YR (fiscal year) PP_FIRST (fund code and

Menu	Menu option	Menuopt file	Accesses ...
			description) PP_SECOND (fund code and description) PP_THIRD (fund code and description) PP_FOURTH (fund code and description) PP_FIFTH (fund code and description) PP_SIXTH (fund code and description) PP_SEVENTH (fund code and description)
	Chart of Accounts	$CARSPATH/menuopt/accounting/others/chartacct	Report Parameters passed: PP_FS_YR (fiscal year) PP_FUND_RAN (fund code range) PP_SORT_FIELD_PRIM (primary sort sequence) PP_SORT_FIELD_SEC (secondary sort sequence)
	Document History	$CARSPATH/menuopt/accounting/others/dochist	Report Parameters passed: PP_FUND (fund code) PP_FUNC (function code) PP_OBJ (object code) PP_SUBFUND (subfund) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_TOTAL (totals only on summary report)
	Document Register	$CARSPATH/menuopt/accounting/reports/docreg	Report Parameters passed: PP_DOC (document code) PP_DOC_NO_BEG (beginning document number) PP_DOC_NO_END (ending document number)
	Interfund Analysis	$CARSPATH/menuopt/accounting/others/duetofrom	Report Parameters passed: PP_FS_CODE (fiscal year period code) PP_FS_YR (fiscal year)
	Journals by Date	$CARSPATH/menuopt/accounting/reports/vchbydate	Report Parameters passed: PP_DATE_BEG (beginning date) PP_DATE_END (ending date)
	Journals by	$CARSPATH/menuopt/	Report

Menu	Menu option	Menuopt file	Accesses ...
	Type	accounting/reports/ vchbytype	Parameters passed: PP_TVCH (journal type) PP_BEG_VCH (beginning journal number) PP_END_VCH (ending journal number)
Accounting: Subfund Reports menu	Summary	$CARSPATH/menuopt/ accounting/others/ prjsum	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Summary by Funds	$CARSPATH/menuopt/ accounting/others/ prjsumfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP

Menu	Menu option	Menuopt file	Accesses ...
			(subtotaling by block or group) PP_SUMMARY (summary report)
	Trial Balance	$CARSPATH/menuopt/accounting/others/prjtrialbl	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Trial Balance by Funds	$CARSPATH/menuopt/accounting/others/prjtrial.f	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Object Detail	$CARSPATH/menuopt/accounting/others/prjacctdtl	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence)

Menu	Menu option	Menuopt file	Accesses ...
			PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group)
	Object Detail by Funds	$CARSPATH/menuopt/accounting/others/prjacctdfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group)
	Object Detail by Month	$CARSPATH/menuopt/accounting/others/prjacctmon	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function

Menu	Menu option	Menuopt file	Accesses ...
			code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Object Summary	$CARSPATH/menuopt/ accounting/others/ prjacctsum	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)
	Object Summary by Funds	$CARSPATH/menuopt/ accounting/others/ prjacctsfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range)

Menu	Menu option	Menuopt file	Accesses ...
			PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUBT_SCH (subtotaling by schedule) PP_SUMMARY (summary report)
	Object Transactions	$CARSPATH/menuopt/accounting/others/prjaccttr	Report Parameters passed: PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRAN (detail or summary transactions) PP_SUBT_PRGLS (print subtotals by function/subfund)
	Function Detail	$CARSPATH/menuopt/accounting/others/prjcntrdtl	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range)

Menu	Menu option	Menuopt file	Accesses ...
			PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group)
	Function Detail by Funds	$CARSPATH/menuopt/ accounting/others/ prjcntrdfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_SCHGRP (subtotaling by schedule or group)
	Function Detail by Month	$CARSPATH/menuopt/ accounting/others/ prjcntrmon	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUMMARY (summary report)
	Function	$CARSPATH/menuopt/	Report

Menu	Menu option	Menuopt file	Accesses ...
	Summary	accounting/others/ prjcntrsum	Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Function Summary by Funds	$CARSPATH/menuopt/ accounting/others/ prjcntrsfd	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)
	Function Transactions	$CARSPATH/menuopt/ accounting/others/ prjcntrtr	Report Parameters passed: PP_SORT_FIELD (sort

Menu	Menu option	Menuopt file	Accesses ...
			sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRAN (detail or summary transactions)
	Function Detail/Trans	$CARSPATH/menuopt/ accounting/scripts/ sortproj	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_SORT_FIELD (sort sequence) PP_PERSON (responsible person) PP_FS_RAN (fiscal period range) PP_FS_YR (fiscal year) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_TRAN (detail or summary transactions) PP_SUBT_SCHGRP (subtotaling by schedule or group)
Accounting: Subsidiary Reports menu	AC Cash by Total Code	$CARSPATH/menuopt/ accounting/reports/ subtcash	Report Parameters passed: PP_NUM_BEG (beginning journal number) PP_NUM_END (ending journal number)

Menu	Menu option	Menuopt file	Accesses ...
	S/L Account Balances	$CARSPATH/menuopt/accounting/reports/subbalance	Report Parameters passed: PP_SUBS (subsidiary code) PP_DATE (balance date) PP_BAL_DEBIT (include debit balances) PP_BAL_CREDIT (include credit balances) PP_BAL_ZERO (include zero balances) PP_AMT (amount types to include) PP_SUBI (additional subsidiary information) PP_SUBPROG (subprogram code)
	S/L Balances by S/L Period	$CARSPATH/menuopt/accounting/reports/subbalprds	Report Parameters passed: PP_SUBS (subsidiary code) PP_DATE (balance date) PP_FIRST (first period for column totals) PP_SECOND (second period for column totals) PP_THIRD (third period for column totals) PP_FOURTH (fourth period for column totals) PP_FIFTH (fifth period for column totals) PP_SIXTH (sixth period for column totals) PP_BAL_DEBIT (include debit balances) PP_BAL_CREDIT (include credit balances) PP_BAL_ZERO (include zero balances) PP_SUBPROG (subprogram code)
	S/L Cash Entries	$CARSPATH/menuopt/accounting/reports/subentcash	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_DATE_BEG (beginning date) PP_DATE_END (ending date) PP_ENT (entry type code) PP_AMT (amount type) PP_AMOUNT (comparison amount) PP_SUMMARY (summary

Menu	Menu option	Menuopt file	Accesses ...
			report)
	S/L Entries by Date	$CARSPATH/menuopt/ accounting/reports/ subentdate	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_DATE_BEG (beginning date) PP_DATE_END (ending date) PP_SUBB (subsidiary balance code) PP_ENT (entry type code) PP_AMT (amount type) PP_SUMMARY (summary report)
	S/L Entries by G/L Period	$CARSPATH/menuopt/ accounting/reports/ subentprds	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_FS_RAN (fiscal period codes) PP_FS_YR (fiscal year) PP_SUBB (subsidiary balance code) PP_ENT (entry type code) PP_AMT (amount type) PP_SUMMARY (summary report)
	S/L Entries by S/L Period	$CARSPATH/menuopt/ accounting/ reports/ subentprd	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_SUBP (subsidiary balance period code) PP_FS_YR (fiscal year) PP_SUBB (subsidiary balance code) PP_ENT (entry type code) PP_SUMMARY (summary report)
	S/L Totals by Payment	$CARSPATH/menuopt/ accounting/reports/ subtbypmt	Report Parameters passed: PP_ID (subsidiary number (ID)) PP_CODE (payment form code) PP_NUM (payment form number)
	S/L Transactions by Date	$CARSPATH/menuopt/ accounting/reports/ subtrdate	Report Parameters passed: PP_SUBS (subsidiary code)

Menu	Menu option	Menuopt file	Accesses ...
			PP_ID (subsidiary number (ID)) PP_DATE_BEG (beginning report date) PP_DATE_END (ending report date) PP_SUBB (subsidiary balance code) PP_SUBT (subsidiary total code) PP_ENT (entry type) PP_PG_BRK (page break flag)
	S/L Transactions by Entry	$CARSPATH/menuopt/accounting/reports/subtrent	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_SUBP (balance period) PP_FS_YR (fiscal year) PP_SUBB (subsidiary balance code) PP_ENT (entry type)
	S/L Transactions by Total	$CARSPATH/menuopt/accounting/reports/subtrtot	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_SUBP (balance period) PP_FS_YR (fiscal year) PP_SUBB (subsidiary balance code) PP_SUBT (subsidiary total code)
	Student Charges/ Payments	$CARSPATH/menuopt/stubill/reports/pdchgtot	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_SUBP (balance period) PP_SUBB (subsidiary balance code) PP_SUBS_POST (posting category)
	Subsidiary History	$CARSPATH/menuopt/accounting/reports/subtrhist	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_DATE_BEG (beginning report date) PP_DATE_END (ending report date)
	Total	$CARSPATH/menuopt/	Report

Menu	Menu option	Menuopt file	Accesses ...
	Balances/GL Account	accounting/others/ totbal	Parameters passed: PP_FS_RAN (fiscal period code range) PP_FS_YR (fiscal year) PP_SORT_FIELD_PRIM (primary sort column) PP_SORT_FIELD_SEC (secondary sort column) PP_SUBS_POST (posting category)
	Total Balances/ Person	$CARSPATH/menuopt/ accounting/reports/ substot	Report Parameters passed: PP_SUBS (subsidiary code) PP_SUBP (balance period) PP_FS_YR (fiscal year) PP_SUBT (subsidiary total code) PP_ID (subsidiary number (ID)) PP_SUBS_POST (posting category)
Accounting: General Ledger Maintenance menu	S/L Balance Status Report	$CARSPATH/menuopt/ accounting/programs/ sbst.sr	Report Parameters passed: PP_SUBS (subsidiary)
	S/L Balance Status Update	$CARSPATH/menuopt/ accounting/programs/ sbst.sru	Report Parameters passed: PP_SUBS (subsidiary)
	Terminate G/L Accounts	$CARSPATH/menuopt/ accounting/informers/ termacct	Script Parameters passed: PP_FS_YR (fiscal year) PP_FUND (fund code) PP_FUNC (function code) PP_OBJ (object code) PP_SUBFUND (subfund code)
	Update G/L Accounts	$CARSPATH/menuopt/ accounting/screens/ glacct	PERFORM screen: General Ledger Account record
	Update G/L Descriptions	$CARSPATH/menuopt/ accounting/scripts/ updgla	Csh script: updgla
	FASB 117 G/L Mapping Rpt	$CARSPATH/menuopt/ accounting/reports/ fasbgla	Report Parameters passed: PP_FS_YR (fiscal year)
	G/L Net Asset Excptn Rpt	$CARSPATH/menuopt/ accounting/reports/ exntast	Report Parameters passed: PP_FS_YR (fiscal year)
Accounting: Audit menu	G/L Report	$CARSPATH/menuopt/ accounting/programs/ glau	Program: *glaudit* Parameters passed: -y (fiscal year) -t (account type) -m (period code) -u (update flag)

Menu	Menu option	Menuopt file	Accesses ...
			-n (mail notification) -c (mail report copy) -d (audit flag for cash balance in doc_table)
	G/L Report and Update	$CARSPATH/menuopt/accounting/programs/glau.u	<u>Program</u>: *glaudit* <u>Parameters passed</u>: -y (fiscal year) -t (account type) -m (period code) -u (update flag) -n (mail notification) -c (mail report copy) -d (audit flag for cash balance in doc_table)
	S/L Report	$CARSPATH/menuopt/accounting/programs/saau.sp	<u>Program</u>: *saaudit* <u>Parameters passed</u>: -s (subsidiary code) -p (pass flag; 1=validate subsidiary, 2=validate control account, 3=validate both) -I (ID number) -l (limited pass 1 verification (does not verify bals or tots) -b (beginning date for verifying pass 2) -e (ending date for verifying pass 2) -m (send mail to user)
	S/L Report and Update	$CARSPATH/menuopt/accounting/programs/saau.u	<u>Program</u>: *saaudit* <u>Parameters passed</u>: -s (subsidiary code) -p (pass flag; 1=validate subsidiary, 2=validate control account, 3=validate both) -I (ID number) -l (limited pass 1 verification (does not verify bals or tots) -b (beginning date for verifying pass 2) -e (ending date for verifying pass 2) -m (send mail to user)
	S/L Report by ID	$CARSPATH/menuopt/accounting/programs/saau.i	<u>Program</u>: *saaudit* <u>Parameters passed</u>: -s (subsidiary code) -p (pass flag; 1=validate subsidiary, 2=validate control account, 3=validate both) -I (ID number) -l (limited pass 1 verification (does not verify bals or tots) -b (beginning date for verifying pass 2) -e (ending date for verifying

Menu	Menu option	Menuopt file	Accesses ...
			pass 2) -m (send mail to user)
	S/L Report by ID/ Update	$CARSPATH/menuopt/ accounting/programs/ saau.iu	Program: *saaudit* Parameters passed: -s (subsidiary code) -p (pass flag; 1=validate subsidiary, 2=validate control account, 3=validate both) -I (ID number) -l (limited pass 1 verification (does not verify bals or tots) -b (beginning date for verifying pass 2) -e (ending date for verifying pass 2) -m (send mail to user)
	G/L Control Account Report	$CARSPATH/menuopt/ accounting/programs/ saau.p2	Program: *saaudit* Parameters passed: -s (subsidiary code) -p (pass flag; 1=validate subsidiary, 2=validate control account, 3=validate both) -I (ID number) -l (limited pass 1 verification (does not verify bals or tots) -b (beginning date for verifying pass 2) -e (ending date for verifying pass 2) -m (send mail to user)
Accounting: Journal Processing menu	Accounting Entry	$CARSPATH/menuopt/ accounting/screens/ voucher	Program: *voucher*
	Void Documents	$CARSPATH/menuopt/ accounting/programs/ docv	Program: *docvoid* Parameters passed: -d (posting date)
	G/L Journal Reports	$CARSPATH/menuopt/ accounting/scripts/ jrnlgl.AC	Csh script: jrnlgl.AC Parameters passed: PP_TVCH (journal code) PP_BEG_VCH_NO (beginning journal number) PP_VCH_END (ending journal number) PP_OUTPUT (printer name)
	S/L Journal Reports	$CARSPATH/menuopt/ accounting/scripts/ jrnlsl.AC	Csh script: jrnlsl.AC Parameters passed: PP_TVCH (journal code) PP_BEG_VCH_NO (beginning journal number) PP_VCH_END (ending journal number) PP_OUTPUT (printer name)
Accounting: ASCII Posting	Post ASCII Files to G/L	$CARSPATH/menuopt/ accounting/programs/	Program: *filepost*

Menu	Menu option	Menuopt file	Accesses ...
menu		fpst.im	
Accounting: Journal Maintenance menu	Journal Recovery	$CARSPATH/menuopt/ accounting/programs/ vchr	Program: *vchrecover*
	Update Journal Records	$CARSPATH/menuopt/ accounting/screens/ voucher	Program: *voucher*
	Finish a Journal	$CARSPATH/menuopt/ accounting/programs/ fpst.f	Program: *filepost*
	Terminate a Journal	$CARSPATH/menuopt/ accounting/programs/ fpst.t	Program: *filepost*
	Post to General Ledger	$CARSPATH/menuopt/ accounting/programs/ fpst.i	Program: *filepost*
Accounting: Recurring Journal Entry menu	Recurring Journal Entries	$CARSPATH/menuopt/ accounting/programs/ recur	Program: *recurent* Parameters passed: -t (table permissions) -a (G/L account permissions) -p (posting permissions) -r (journal reference)
	Recurring Jrnl Entry Rpt	$CARSPATH/menuopt/ accounting/reports/ recur	Report Parameters passed: -f (formtype)
	Recur Table Report	$CARSPATH/menuopt/ accounting/reports/ trecur	Report Parameters passed: -f (formtype)
Accounting: Standard Accounting Entries menu	Table	$CARSPATH/menuopt/ accounting/screens/ sae	PERFORM screen: sae_table
	Table Report	$CARSPATH/menuopt/ accounting/reports/ saelist	Report Parameters passed: -f (formtype)
	Verify Standard Entries	$CARSPATH/menuopt/ accounting/programs/ sae.rt	Program: *sae* Parameters passed: -m (month) -y (fiscal year) -d (effective date for entry) -c (entry code) -r (report flag) -t (test flag)
	Create Standard Entries	$CARSPATH/menuopt/ accounting/programs/ sae.r	Program: *sae* Parameters passed: -m (month) -y (fiscal year) -d (effective date for entry) -c (entry code) -r (report flag) -t (test flag)
	Post to	$CARSPATH/menuopt/	Program: *filepost*

Menu	Menu option	Menuopt file	Accesses ...
	General Ledger	accounting/programs/ fpst.iAC	
Accounting: Period End Processing: Form 990 Reports menu	IRS Donor List	$CARSPATH/menuopt/ accounting/scripts 990donor	Report Parameters passed: PP_DATE_BEG (beginning date) PP_DATE_END (ending date) PP_AMOUNT (minimum total giving amount) PP_AMOUNT (minimum amount for single gifts)
	Schedule A - Part I	$CARSPATH/menuopt/ accounting/scripts 990A1	Report Parameters passed: PP_DATE_BEG (beginning date) PP_DATE_END (ending date) PP_AMOUNT (comparison amount) PP_DEDUCTION (deduction code that determines excluded benefits) PP_RUNCODE (runcode for the report)
	Schedule A - Part II	$CARSPATH/menuopt/ accounting/scripts 990A2	PP_DATE_BEG (beginning date) PP_DATE_END (ending date) PP_AMOUNT (comparison amount)
Accounting: Period End Processing: General Ledger Closing menu	Trial Balance	$CARSPATH/menuopt/ accounting/others/ trialbal	Report Parameters passed: PP_COL_OUTPUT_TYPE (formtype) PP_FS_RAN (fiscal period range) PP_AMT (amount type) PP_FUND_RAN (fund code range) PP_FUNC_RAN (function code range) PP_OBJ_RAN (object code range) PP_SUBFUND_RAN (subfund range) PP_NONDSPL_OBJ (nondisplay of selected objects) PP_SUBT_BLGRP (subtotaling by block or group) PP_SUMMARY (summary report)

Menu	Menu option	Menuopt file	Accesses ...
	Add G/L Accounts	$CARSPATH/menuopt/ accounting/informers/ addgla	<u>SQL script</u> <u>Parameters passed</u>: PP_FISCAL_YR (fiscal years to copy from and to)
	Add Closing Fund Balances	$CARSPATH/menuopt/ accounting/informers/ addclsgfb	<u>SQL script</u> <u>Parameters passed</u>: PP_FS_YR (fiscal year) PP_FUND_RAN (fund code range) PP_FUND_BAL (fund balance) PP_OBJ_RAN (object code range) PP_AMT (amount type)
	Edit Closing Fund Balances	$CARSPATH/menuopt/ accounting/screens/ clsgfb	<u>PERFORM screen</u>: Closing Fund Balance Record
	Closing Fund Bal Report	$CARSPATH/menuopt/ accounting/reports/ clsgfbacct	<u>Report</u> <u>Parameters passed</u>: PP_FS_YR (fiscal year) PP_FUND_RAN (fund code range)
	Missing Fund Bal Report	$CARSPATH/menuopt/ accounting/reports/ clsgfbmiss	<u>Report</u> <u>Parameters passed</u>: PP_FS_YR (fiscal year) PP_FUND_RAN (fund code range)
	Edit Check for Closing	$CARSPATH/menuopt/ accounting/programs/ glcled	<u>Program</u>: *glclcked* <u>Parameters passed</u>: -f (fund account) -y (fiscal year)
	Create Closing Entry	$CARSPATH/menuopt/ accounting/programs/ glcl	<u>Program</u>: *glclsg* <u>Parameters passed</u>: -y (fiscal year) -t (amount type) -f (fund code) -v (journal)
	Create Balance Forward Ent	$CARSPATH/menuopt/ accounting/programs/ glbf	<u>Program</u>: *glbalfwd* <u>Parameters passed</u>: -y (fiscal year) -t (amount type) -f (fund code) -v (journal reference code)
	Post to General Ledger	$CARSPATH/menuopt/ accounting/programs/ fpst.iACPC	<u>Program</u>: *filepost*
	G/L Journal Reports	$CARSPATH/menuopt/ accounting/scripts/ jrnlg.ACPC	<u>Script</u> jrnlg.ACPC <u>Parameters passed</u>: PP_TVCH (journal code) PP_BEG_VCH_NO (beginning journal number) PP_VCH_END (ending journal number) PP_OUTPUT (printer)

Menu	Menu option	Menuopt file	Accesses ...
Accounting: Period End Processing: Subsidiary Archiving menu	Subsidiary History Report	$CARSPATH/menuopt/ accounting/reports/ subtrh.A	Report Parameters passed: PP_SUBS (subsidiary code) PP_ID (subsidiary number (ID)) PP_DATE_BEG (beginning date) PP_DATE_END (ending date)
	Verify Archival Data by ID	$CARSPATH/menuopt/ accounting/programs/ sarc.beo	Program: *sarc* Parameters passed: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries) -j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)
	Verify Archival Data	$CARSPATH/menuopt/ accounting/programs/ sarc.o	Program: *sarc* Parameters passed: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries) -j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)
	Verify All S/L Data by ID	$CARSPATH/menuopt/ accounting/programs/ sarc.beoj	Program: *sarc* Parameters passed: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries) -j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)

Menu	Menu option	Menuopt file	Accesses ...
	Verify All S/L Data	$CARSPATH/menuopt/ accounting/programs/ sarc.oj	<u>Program</u>: *sarc* <u>Parameters passed</u>: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries) -j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)
	Mark Data for Archiving/ID	$CARSPATH/menuopt/ accounting/programs/ sarc.bueo	<u>Program</u>: *sarc* <u>Parameters passed</u>: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries) -j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)
	Mark Data for Archiving	$CARSPATH/menuopt/ accounting/programs/ sarc.uo	<u>Program</u>: *sarc* <u>Parameters passed</u>: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries) -j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)
	Remove Archival Data	$CARSPATH/menuopt/ accounting/programs/ sarc.r	<u>Program</u>: *sarc* <u>Parameters passed</u>: -s (subsidiary code) -d (archive date) -u (update flag) -b (beginning ID number) -e (ending ID number) -o (output summary report of archived entries)

Menu	Menu option	Menuopt file	Accesses ...
			-j (output summary report of non-archived entries) -r (delete records from database flag) -v (turn on debugging output flag)
Accounting: Period End Processing: Subsidiary Balance Forward menu	Credit Balances Forward	$CARSPATH/menuopt/ accounting/programs/ sabf.g	Program: *sabalfwd* Parameters passed: -s (subsidiary code) -b (subsidiary balance code) -c (subsidiary total code) -f (list of from sessions) -t (target session) -I (list of ID numbers)
	Credit Balances Forward / ID	$CARSPATH/menuopt/ accounting/programs/ sabf.i	Program: *sabalfwd* Parameters passed: -s (subsidiary code) -b (subsidiary balance code) -c (subsidiary total code) -f (list of from sessions) -t (target session) -I (list of ID numbers)
	All Balances Forward	$CARSPATH/menuopt/ accounting/programs/ sabf.dg	Program: *sabalfwd* Parameters passed: -s (subsidiary code) -b (subsidiary balance code) -c (subsidiary total code) -f (list of from sessions) -t (target session) -I (list of ID numbers)
	All Balances Forward by ID	$CARSPATH/menuopt/ accounting/programs/ sabf.di	Program: *sabalfwd* Parameters passed: -s (subsidiary code) -b (subsidiary balance code) -c (subsidiary total code) -f (list of from sessions) -t (target session) -I (list of ID numbers)
	Post to General Ledger	$CARSPATH/menuopt/ accounting/programs/ fpst.iAC	Program: *filepost*
	G/L Journal Reports	$CARSPATH/menuopt/ accounting/scripts/ jrnlgl.AC	Script jrnlg.AC Parameters passed: PP_TVCH (journal code) PP_BEG_VCH_NO (beginning journal number) PP_VCH_END (ending journal number) PP_OUTPUT (printer)
	S/L Journal Reports	$CARSPATH/menuopt/ accounting/scripts/ jrnlsl.AC	Script jrnlsl.AC Parameters passed: PP_TVCH (journal code) PP_BEG_VCH_NO (beginning journal number)

Menu	Menu option	Menuopt file	Accesses ...
			PP_VCH_END (ending journal number) PP_OUTPUT (printer)
	S/L Balance Status Report	$CARSPATH/menuopt/accounting/programs/sbst.sr	Program: *subbstat* Parameters passed: -s (subsidiary code)
	S/L Balance Status Update	$CARSPATH/menuopt/accounting/programs/sbst.sru	Program: *subbstat* Parameters passed: -s (subsidiary code) -u (update balance flag)

PERFORM (Table Maintenance) Screens

Introduction

General Ledger uses PERFORM screens for displaying tables and some records. You can access the screen files in the following directory path:
$CARSPATH/modules/accounting/screens

PERFORM screens

The following list contains the PERFORM screens used in General Ledger:

Screen file: clsgfb
Screen title: Closing Fund Balance Record

Screen file: cntrcomb
Screen title: Combined Function Table

Screen file: fiscalcal
Screen title: Fiscal Calendar

Screen file: glacct
Screen title: General Ledger Account Records

Screen file: glas
Screen title: General Ledger Association Table

Screen file: glatemp
Screen title: Temporary G/L Records

Screen file: gldefine
Screen title: Defined Account Table

Screen file: glfieldrpt
Screen title: Chart of Accounts Tables

Screen file: sae
Screen title: Standard Accounting Entry Table/Standard Accounting Entry Records

Screen file: tatype
Screen title: Amount Type Table

Screen file: tclaim
Screen title: Claim Table

Screen file: tent
Screen title: Entry Type Table

Screen file: tfinset
Screen title: Financial Statement Set Table

Screen file: tfs
Screen title: Financial Statement Table

Screen file: tglperm
Screen title: General Ledger Permission Table

Screen file: tglsub
Screen title: G/L Account Auto-Fill Table

Screen file: tsubs
Screen title: Subsidiary Table

Screen file: tsubtbls
Screen title: Subsidiary Balance/Total/Association Table

Screen file: tvch
Screen title: Journal Type Table

Screen file: voucher
Screen title: Journal Records

General Ledger SQL Scripts

Introduction

The General Ledger module contains SQL scripts that directly access the database, performing queries and updating the database. The scripts are located in the following directory path: $CARSPATH/modules/accounting/informers

Note: Reports can also create SQL scripts, in which the reports print the statements they use to execute. One purpose of these reports is to provide an efficient way for SQL to update a table with the contents of another table.

In addition, Csh scripts can call SQL scripts and ACE reports. Such ACE reports and SQL scripts do not reside on the CX menu system.

SQL scripts

The following SQL scripts relate to General Ledger.

Menu option: Add Closing Fund Balances
SQL script: addclsgfb
Description: Creates Closing Fund Balance records for the General Ledger Closing process. Closing Fund Balance records define the accounts to be closed, and the fund balance account into which to close the accounts.
Tables used:
- General Ledger Amount record (gl_amt_rec)
- Closing Fund Balance records (clsgfb_rec)

Menu option: Add G/L Accounts
SQL script: addgla
Description: Creates General Ledger Account records for the new fiscal year for all accounts that are active in the previous fiscal year.

Note: Before running this script, set the Terminate flag on accounts in the old fiscal year that you do not want to use in the new year.

Tables used:
- General Ledger Account record (gla_rec)

Menu option: Initialize Report Indexes
SQL script: addglatemp
Description: Creates temporary records that include all active General Ledger accounts for two fiscal years (the year specified, and the year previous to the specified year). The temporary records enable the financial reports to run at maximum efficiency.

CAUTION: You must run this option for the correct fiscal year before running an accounting report with optional column output. If you do not run this option, the amounts on the reports may not be accurate.

Tables used:
- General Ledger Account record (gla_rec)
- General Ledger Amount record (gl_amt_rec)
- General Ledger Temporary record (glatemp_rec)

Menu option: General Ledger Maintenance menu: Terminate G/L Accounts
SQL script: termacct
Description: Sets the Terminate flag on the General Ledger account passed.

Note: Run this option for each year for which you want to terminate accounts.

Tables used:

- General Ledger Account record

General Ledger Csh Scripts

Introduction

General Ledger contains Csh scripts to automate the processing of information. Csh scripts are UNIX-based program statements that can execute a series of SQL scripts or reports.

All the Csh scripts that relate to General Ledger are database related, updating or reporting database information. Some Csh scripts in the directory location $CARSPATH/modules/common/scripts are non-database related, and perform maintenance procedures.

The General Ledger Csh scripts are located in the following directory path: $CARSPATH/modules/accounting/scripts.

Csh Scripts

The following list associates a General Ledger menu option with the corresponding Csh script and provides a description of the script.

Note: In the following list, descriptions of Csh scripts include:

- Purpose of the script
- A list of SQL statements used, if applicable

Menu option: Combined Function Reports
Csh script: cntrdtlcom
Purpose: Prints multiple reports

Menu option: Add Fiscal Calendar (under Table Maintenance)
Csh script: fiscalnxt
Purpose: Creates a new set of Fiscal Calendar records for the next fiscal year, based on the values in the current year's records

Note: The script *fiscalnxt* runs an ACE report that outputs SQL statements, then executes the output statements.

Menu option: General Ledger Report
Csh script: glfieldrpt
Purpose: Produces a single account-related table

Menu option: Function Reports: Detail/Transactions
Csh script: sortcntr
Purpose: Merges detail and transaction function reports for departmental distribution

Menu option: Subfund Reports menu
Csh script: sortproj
Purpose: Merges detail and transaction subfund reports for departmental distribution

Menu option: General Ledger Maintenance Menu
Csh script: updgla
Purpose: Updates account descriptions in gla_recs

Note: The script *updgla* runs an ACE report that outputs SQL statements, then executes the output statements.

Menu option: Accounting: Reports menu
Csh script: jrnlgl
Purpose: Prints multiple General Ledger reports

Menu option: Accounting: Subsidiary Reports Menu

Csh script: jrnlsl
Purpose: Prints Subsidiary Ledger reports

INDEX