NewCash User Manual

Don Allen

August 31, 2019

Contents

List of Figures

Chapter 1

Motivation

When I began work on Newcash in the summer of 2012, I had had long experience with financial management software, having begun managing my finances on a MacIntosh in 1988. Since that time, I've used a number of personal finance applications. Prior to deciding to build my own system, I used the Linux version of Gnucash, arguably the most popular of the open-source financial packages.

While I was generally pleased with Gnucash during the 8 years I used it, the reporting subsystem was a weak spot for me. I consider good financial reports a primary benefit of tracking finances with a computer, and I could not get the reports I wanted from Gnucash in the form I wanted them. A secondary issue was the slowness of the reporting subsystem, but if I could have gotten the results I wanted, I would have accepted this.

Dissatisfied with Gnucash's reports, I wrote my own report-generating program, which read my financial data from the Gnucash database. As I will discuss later, Gnucash was originally designed to store its data in a specially formatted file (the format is called XML), not in a database. Some years ago, a new back-end was added to provide the option of storing Gnucash data using one of several database systems. My report generator depended on my Gnucash data residing in a database because generating reports from the original file format would have been more difficult.

This method worked reasonably well for a time, until I learned, while reading the Gnucash developers' mailing list (a mailing list not typically read by Gnucash users; it is used mostly for the developers to talk to each other; there is another mailing list for users to discuss issues with each other and with developers), that the developers were aware of data-loss issues with the database back-end. This obviously made it necessary for me to immediately switch back to storing my (20 years of) financial data in the original XML format. To obtain reports, I now had to copy my data to a temporary database for use by my report generator, a cumbersome process. Inconvenience aside, what really concerned me was that there was no formal announcement of the data-loss issue to the user community, and no withdrawal of the database back-end until the problems were fixed. I began to investigate personal finance managers other

than Gnucash.

I took a look at KMyMoney, another open-source financial application, but its Gnucash data importer failed, causing the program to crash, and I was not able to find a workaround. I was not about to switch back to Quicken, which had become conceptually messy bloatware and which would have required either running Windows, or paying Apple's hardware premium for a Mac, or building a Hackintosh. Microsoft Money had died a well-deserved death. With no good existing choices and armed with almost 50 years of experience as a software developer, and 24 years of experience as a user of financial management software, I decided to build my own system.

Chapter 2

Design

Newcash was written by me for me. Its design reflects my requirements and opinions about accounting, software implementation, and user interfaces. I am sharing it because my computing life has been greatly enhanced by the existence of open-source/free software, and it seems appropriate to give back to the open-source community, now that I have something to offer that community in addition to financial support. But do understand that this software is appropriate only for people familiar with and comfortable with Unix/Linux, aren't frightened by the prospect of running a shell script (and perhaps even modifying one for their own use) and remember a bit of grade-school arithmetic (if you can do cash-flow problems with an HP 12C business calculator, you qualify!). This software is not for people whose idea of a computer is an iPad.

An additional design objective for Newcash was to heed the words of Albert Einstein: "Things should be as simple as possible, but not simpler."

As a result of these considerations, Newcash

- is designed to run only on Linux and Unix[1].

- data is stored only in a Sqlite database. Sqlite is a widely used open-source database system (Firefox uses it, for example) that has a simple

---

[1]Linux is a clone of Unix. By "Unix" I mean the descendents of the Berkeley Software Distribution: FreeBSD, OpenBSD, NetBSD, DragonFlyBSD, and the Mac OS. Unix was originally developed at Bell Labs in the late 1960s. The University of California Berkeley Computer Science department modified the Bell Labs system, creating a version that found widespread popularity in the 1970s and 1980s. In the early 1990s, just when PC hardware became powerful enough to support Unix, AT&T sued the Regents of the University of California, as well as a company formed by former members of the Berkeley CS department, which tied up the Unix source code in the courts at a crucial time. This led to the development of the Linux kernel (the kernel of an operating system is the software that manages the hardware itself and provides fundamental services to software layered above it) by Linus Torvalds, a native of Finland who was a computer science graduate student at the time. Linux is a complete rewrite of Unix, thereby avoiding the pointless legal hassle. The lawsuit was eventually settled out of court, but by then, Linux had established a toe-hold on PC hardware that it has never relinquished and Torvalds has become deservedly famous in computing circles. He continues to devote himself to the further development of the Linux kernel.

architecture that eliminates the separate database server found in many other database systems. This has an important consequence: applications can make use of Sqlite without any system- or database-administration effort on the part of the end user. Furthermore, Sqlite is fine work, very robust and very fast. I decided at the outset of my work on Newcash that I would take full advantage of Sqlite's capabilities. One of the reasons for Newcash's small size, in terms of number of lines of code, is attributable to this decision.

- supports only a single language (English).

- supports only a single currency, about which it is unspecific. It can be used to manage finances in *any* single currency, so long as that currency is used consistently throughout.

- is designed for managing *personal* finances only; there is no support for small business accounting.

- was designed to be fast and efficient, both in terms of startup time, execution speed and ease of the user's workflow. The extent to which I have succeeded in achieving this is due largely, in my opinion, to the fact that I began managing my own finances with Newcash as soon as it was up and limping in January, 2013. Nothing focuses the mind of a software developer more effectively than having to use his own product, especially when his money is involved.

  Another factor was the availability of a good open-source database, Sqlite. Newcash was designed around that database and uses it as its main data repository. Newcash does not access any data in the database that isn't needed either for display or for handling a user's interaction. Contrast with Gnucash, whose developers did not have the luxury of a Sqlite available to them in the mid-1990s when they began work on their system. So, as mentioned earlier, Gnucash was originally designed to store its data in an XML file, which meant (and means to this day) that entire file, all the user's data, is loaded into memory at startup. For some, perhaps many, that won't matter, because their data files are small. But I was managing over 20 years of financial data with Gnucash, which made its startup time annoying slow, measured in minutes (Newcash starts essentially instantaneously regardless of how much data the user has, because, again, it only accesses data that is actually needed to satisfy the user's requests). Their database backend is used the same way, loading the entire database into memory at startup, because that's the way the program was designed. Had I written Newcash in the late 1990s, I probably would have done the same thing; the original Gnucash developers didn't have the same choices available to them that I had 15 years later, when work on Newcash began.

- employs a simplified approach to double-entry accounting.

The original version of Newcash was written largely in C, with some utilities written in Tcl. While I have written a lot of C code over many years, the language leaves a lot to be desired, not surprising since it dates to the late 1960s. When things started to become unwieldly, not as readable as I would like, and an occasional memory-management problem appeared, I began to look for an alternative. Scheme, Go, Haskell and Nim were all considered and rejected for a variety of reasons. I finally settled on Rust. This is not an obvious choice, I will concede, because Rust is intended as a systems language, achieving memory safety without a garbage collector. This makes it one of the most difficult languages to master that I have encountered in my long career as a software developer. But I have found a subset of Rust that works well for me. Rust's toolset is excellent, it offers good libraries for using Sqlite and Gtk3, and the performance of the programs is indistinguishable from the C versions. But I regard the two main virtues of the re-write as far more readable code and the greater correctness brought about by the rigor demanded by the Rust compiler. I have required of myself that all of the subsystems of Newcash written in Rust compile without error or warning.

As of this writing, Newcash consists of

- 9,573 lines of Rust (this includes Newcash, several utilities including the report generator, as well as the SQL I use in the system),

- 1,908 lines of scripts, mostly Tcl, the rest shell scripts (the transaction schedulers, the database creator and converter, the program for obtaining security quotes, and other useful utilities).

- an 8-row Libreoffice spreadsheet, for calculating mortgage payments

for a total of 11,489 lines of code.
By way of comparison, Gnucash 3.6 consists of

- 477411 lines of C,

- 68512 lines of Scheme (reports),

- 12587 lines of Python

- 230 lines of Perl

for a total of 558,740 lines of code in 4 programming languages. The Gnucash developers have written almost 49 times as much code as I have. Gnucash does more than Newcash (e.g., it has support for small-business accounting, multi-language support, multi-currency support, multiple database support), but the systems' core personal financial management functionalities are similar.

Chapter 3

Double-Entry Accounting

A key element of any financial software is the accounting system that it employs. Newcash, and before it, Gnucash and Managing Your Money, are/were based on an accounting method known as "double-entry". Double-entry is an ancient accounting system; according to Wikipedia, double-entry accounting "was first codified in the 15th century by the Franciscan friar Luca Pacioli".

Despite its age, double-entry bookkeeping is still widely used in business accounting, but it has ways of presenting financial transactions and their effect on account balances that I believe have outlived their usefulness. I will go into more detail about this below. Gnucash uses the traditional method, but tries to make it more palatable to non-accountants, with modest success. But for me, this is a band-aid; I prefer to deal with the fundamental problem. And so Newcash departs from tradition when presenting financial data. This might upset accountants used to the traditional method of doing things, but I believe the majority of Newcash users will not be professional accountants and will benefit from the simplicity of the method I have chosen.

After presenting some definitions that are basic to any discussion of double-entry accounting, I will devote a subsection to explaining the difference between the traditional approach and the method I've adopted for Newcash. I will provide examples from both systems to illustrate the differences.

3.1 Definitions

Accounts Accounts are objects from which and to which money flows. Every account has a balance, which is the sum of all the account's signed money flows to date. I will have more to say about accounts in Section 4.

Transactions Transactions move money from one set of accounts, the "sources", to another set of accounts, the "destinations". The properties of transactions are the date on which they occurred, an identifying number (such as a check number), and a description. Notice that no amounts are contained

within a transaction itself. Amounts are stored in "splits" that are associated with transactions. A transaction may have many splits associated with it.

Splits Splits are associated with one and only one transaction and with one and only one account. Splits contain a signed money amount or value. If the amount is positive, then the split represents that amount flowing *into* the split's account. If the amount is negative, then the split represents that amount flowing *from* the split's account. As you will see in a moment, Gnucash and Newcash display those amounts differently.

Account Registers Account registers bear some resemblence to the traditional paper check-book register, but they contain more information.

3.2   Why Double-Entry?

Double-entry bookkeeping is based on two fundamental principles:

- In every financial transaction, money flows from (possibly multiple) source accounts to (possibly multiple) destination accounts.

- Regarded as signed quantities, the sum of the flows from the sources to the destinations in every transaction must equal zero.

Let's assume your monthly mortgage payment is $1224.73. If you pay your mortgage and make an entry in your checkbook register, or perhaps a spreadsheet, indicating that you have written a check for $1224.73 to your lender, you've only told part of the story. You've noted that $1224.73 left your checking account, but you didn't say in proper bookkeeping terms where it went. You may have written the name of the lender in the description of the transaction, but how much of the payment was principal? Interest? Escrow? You didn't say. What are the new balances of the mortgage principal, interest and escrow accounts? Even if you are maintaining your register in a spreadsheet, you haven't provided enough information to compute these balances. Double-entry to the rescue.

Using the double-entry method, this transaction might consist of four splits, each describing a money flow from or to a single account (I made up the numbers in the following example for purposes of illustration and I have deliberately not described how inflows and outflows are represented; I defer that discussion to Section 3.3):

- A split stating that $1224.73 flowed out of the account

    `:Assets:Bank Accounts:Joint Checking`

    The colons will be explained shortly, but suffice to say that your accounts form a tree-like structure, from very general near the root to very specific

at the leaves. The colons separate parent nodes from child nodes. This is very similar to computer file-systems, which are also tree structures, where slashes separate parent nodes (directories) from child nodes (directories or files), e.g.,

```
/home/jsa/Photographs/Family/Cats/cute.jpg
```

- A split stating that $391.20 flowed into the account

  ```
  Expenses:Housing:Mortage interest
  ```

- A split stating that $666.86 flowed into the account

  ```
  Liabilites:Mortgage principal
  ```

- A split stating that $166.67 flowed into the account

  ```
  Expenses:Housing:Property taxes
  ```

We are now ready to answer the question "Why Double-Entry?". The requirement that all money flows be entered in every transaction, and that they must balance, is a defense against entering amounts incorrectly. And these fully-described transactions permit the generation of essential (if you are serious about money management) reports about where your money has come from and where it has gone (the Income and Expense Statement) and what you own and what you owe (the Balance Sheet).

3.3 The Traditional and Newcash Methods of Presentation

If you read the Wikipedia articles on double-entry bookkeeping

```
http://en.wikipedia.org/wiki/Double-entry_bookkeeping_system
http://en.wikipedia.org/wiki/Debit_and_credit
```

you will learn that

> Generally speaking, the source account for the transaction is credited (an entry is made on the right side of the account's ledger) and the destination account is debited (an entry is made on the left).

Money flowing *from* an account is a credit, money flowing *to* an account is a debit. This is almost certainly the opposite of what you thought. You and I have both been confused because debits and credits are typically spoken of by organizations (e.g., banks, vendors) from the perspective of their *own* accounts. When they say "we are issuing you a credit in the amount of $5.93" they are

referring to their own account. A credit flowing *from* their account becomes a debit flowing *to* your account.

In that same article, you also learn that there is a complicated system that determines how debits and credits affect the balances of various types of accounts[1]:

> If there is an increase or decrease in one account, there will be an equal decrease or increase in another account. There may be equal increases to both accounts, depending on what kind of accounts they are. There may also be equal decreases to both accounts. Accordingly, the following rules of debit and credit in respect to the various categories of accounts can be obtained. The rules may be summarised as below:
>
> **Assets Accounts** debit increases in assets and credit decreases in assets
>
> **Capital Account** credit increases in capital and debit decreases in capital
>
> **Liabilities Accounts** credit increases in liabilities and debit decreases in liabilities
>
> **Revenues or Incomes Accounts** credit increases in incomes and gains and debit decreases in incomes and gains
>
> **Expenses or Losses Accounts** debit increases in expenses and losses and credit decreases in expenses and losses

Got that? If not, you are part of a large crowd, because this is absurdly and unnecessarily complicated. There are actually two oddities here:

- Debits (money flowing to an account) and credits (money flowing away from an account) are unsigned numbers and therefore have to be presented in separate columns to distinguish them.

- The computation of running balances of accounts and the way debits and credits affect them is not uniform. It is designed to produce positive balances, unless something unusual has happened. While debits and credits are unsigned quantities, balances are signed. But because the effect that debits and credits have on those balances depends on the account type, the signs of the balances mean different things for different account types. For example, a positive balance means that the aggregate or net money flow for an Asset account has been towards that account. A positive balance for a Liability account, however, means that the net money flow has been *away* from that account.

---

[1]Here I have quoted the Wikipedia article verbatim, with no attempt to improve the quality of the writing.

There is also an error in the Wikipedia excerpt above: an increase or decrease in one account does *not* necessarily have to be balanced by a decrease or increase in just one account. The balance can be achieved by activity in many accounts within one transaction.

A much simpler method, the method I have chosen for Newcash, is to represent money flows as signed quantities, with the sign indicating the direction of the flow. With this method, a single column is sufficent because positive inflows are easily distinguished from negative outflows. Running balances are computed for all accounts by simple addition of the signed quantities to the previous balance, and so the sign of the balance of any account indicates whether the net money flow has been toward (positive balance) or away (negative balance) from the account.

Why was the added complication of unsigned debits and credits and the associated set of rules for computing balances adopted for traditional double-entry accounting? I think it has its basis in the fear, perhaps justified, that many people don't understand negative numbers and/or associate them with something bad. After all, if you look up the word "negative" in Merriam-Webster, you find

- harmful or bad; not wanted.

- thinking about the bad qualities of someone or something; thinking that a bad result will happen; not hopeful or optimistic.

There's more to the definition, covering the mathematical definition of the word, but it seems that the worry was that most people wouldn't get that far. So it appears that traditional double-entry accounting was designed to avoid negative numbers because the signs of account balances are manipulated so that all account types are likely to have positive balances. It does this at the expense of complexity, wasted space (two columns for money flows where one would do), and an inconsistent meaning of the signs of balances.

Whether or not traditional double-entry made the right choice for people in general is irrelevant to the design of Newcash. Newcash is for people who can remember their elementary school math, do not fear signed or negative numbers and can understand that "negative" is not necessarily synonymous with "bad".

Elaborating on what was said above, all amounts in Newcash are displayed as a single column of signed numbers instead of two columns of unsigned debits and credits. In fact, there is no concept of debits and credits in Newcash at all; the direction of money flow is indicated by the signs of amounts. Negative amounts indicate money flowing *from* an account, positive amounts indicate money flowing *to* an account[2]. Running balances for all accounts are calculated identically, by arithmetic addition. Thus money flowing to any account will increase its balance (make it more positive), whereas money flowing away from an

---

[2]Users of the classic HP 12C business calculator, especially those who used a 12C to solve cash-flow problems, will be familiar with this.

account will decrease its balance (make it more negative)[3]. All the unnecessary complications associated with debits and credits and how they affect balances are replaced by this simple rule.

This does mean that the signs of the balances of some accounts in Newcash will surprise you at first, since we are breaking with the tradition to which you are accustomed. For example, Income accounts will usually have negative balances, meaning that the net flow has been away from them. Income accounts provide ... income. Money normally flows *from* them.

The same is true of Liability accounts, such as credit card accounts or mortgages. Borrowing or credit means that money first flows from the lender to you, resulting in a negative balance for the account representing that lender. When you make payments on the loan, which is money flowing back to the lender, the balance of the liability account representing the loan becomes less negative (closer to zero) but still negative. When you pay off the loan, the balance of the liability account becomes zero. A liability account will not usually have a positive balance, because that would mean that you had paid the lender more than you owed, something you are unlikely to do intentionally.

A negative balance is only bad if it occurs in an account that ought to have a positive balance, such as a checking account. When you open a checking account, what's the first thing you do? Deposit some money. This is a money flow toward the account, thus positive, resulting in a positive balance. Writing checks are money outflows, negative flows, reducing the balance. If you are careful, you will not allow the balance to become negative, meaning that you have over-drawn the account (unless, of course, your account has protection against this, which amounts to the bank lending you money at credit-card interest rates to cover your over-drafts).

Similarly, Expense accounts will tend to have positive balances, since money typically flows to them. For example, when you pay your electric bill, money will flow from your checking account to the expense account that represents the electricity expense.

You will also need to realize that the outside world does things the traditional way. So if you look at a credit-card statement, a balance of 3872.47 means you owe them that amount. The account balance in Newcash would, of course, be -3872.47.

3.3.1 Net Worth and Cash Flow

There is a quantity called "Net Worth" (synonymous with "Equity", or "Shareholder's Equity" in the case of companies that have issued shares), defined as

---

[3]Think of the "real line", a horizontal line denoting the infinity of real numbers, extending toward minus infinity on the left, toward plus infinity on the right, with zero in the middle. An inflow to an account moves its balance to the right on that line. That is what I mean by "increase", or "more positive". The balance may still be negative, but it is less negative – more to the right on the line – than it was before the inflow occurred. Exactly the same applies to outflows; they move the balance to the left on the real line, "decreasing" the balance, making it "more negative".

the sums of the balances, at a specific moment in time, of all your asset and liability accounts:

$$Assets + Liabilities = Net\_Worth \qquad (3.1)$$

In traditional double-entry accounting, because liability account balances are computed using the opposite sign for money flows than is used for assets, the total liability balance must be negated:

$$Assets - Liabilities = Net\_Worth \qquad (3.2)$$

Note that both methods produce the same answer for Net Worth.

A similar quantity, called Cash Flow, is defined as the total flow to your income and expense accounts over a particular time period.

In Newcash, this is written as

$$Income + Expenses = Cash\_Flow \qquad (3.3)$$

As with the equation for Net Worth, a sign reversal is needed in the traditional equation for cash flow, to compensate for the difference in signs used for the balances of income and expense accounts, and also because traditional double-entry wants positive cash flow to be a Good Thing:

$$Income - Expenses = Cash\_Flow \qquad (3.4)$$

These two equations do *not* produce the same answer! Their signs are opposite, because of the aforementioned manipulation of the signs of account balances in traditional double-entry. In Newcash, if the cash flow, as computed in Equation 3.3, is negative, this is a good thing. It means that the net flow of money to/from your income and expense accounts has been *from* them and (typically) toward your asset and liability accounts, increasing their balances and thus your net worth. For example, when you get paid, money flows from your Salary account (an income account) to, perhaps, your Checking account (an asset account). When you pay your electric bill, money flows from your Checking acocunt (an asset account) to your Electricity account (an expense account). It is the net effect of these kinds of transactions that the Cash Flow equation is measuring.

3.3.2   An Example

To illustrate the difference between traditional/Gnucash double-entry and the Newcash approach, let's look at an example – the mortgage payment discussed earlier. Figure 3.1 shows the checking account register in Gnucash, with Gnucash configured to display the traditional labels above the amount columns (not Gnucash's default configuration). Notice that the debits column is to the left of the credits column. This is the traditional way to display debits and credits, regardless of the type of account. Figure 3.2 shows the same register after the Gnucash preferences have been changed to use their more explanatory labels

(the default configuration). In the rest of the discussion of this example, the figures will show Gnucash in its default configuration, unless explicitly noted.

In these first two figures, you can only see the transactions, not the splits. I'd like to call your attention to two things in these figures:

1. In the Paycheck transaction, we see `Income:Salary`[4] in the cell in the "Transfers" column. To its right, in the Debit or Deposit column, we see the amount 3000.00. Does that apply to the `Income:Salary` account? No, it does not. It applies to the `Checking` account. How do I know that? Because it's a debit, a deposit, money flowing to an account. Now see Figure 3.3, which shows the splits for the Paycheck transaction. Now the 3000.00 amount in the row where `Income:Salary` appears in what was the "Transfer" column has moved to the right-hand column, now labeled "Tot Withdrawal", whereas you can now see that the 3000.00 is a Deposit to `Assets:Checking`. So when you toggle the "Splits" button, you have to change how you associate the account appearing in the column to the left of the amounts with those amounts. When splits are not displayed, it applies to the account for which you have displayed the register. When splits are displayed, it applies to accounts shown in the column to the left of the amounts. In my many years of using Gnucash, I always found this a bit confusing and fixing this user-interface issue played a part in the design of Newcash.

2. In the case of the Mortgage Payment transaction, the "Transfers" column contains "– Split Transaction –", whereas the "Paycheck" transaction has `Income:Salary` in that position. Does this mean that the Mortgage Payment transaction has splits and the "Paycheck" transaction does not? The wording in the "Transfer" column for the "Mortgage Payment" transaction certainly suggests this, but it is not the case. All transactions have splits. The distinction is that the "Mortgage Payment" transaction has more than two splits, and so there isn't a single account that can serve as the one and only "Transfer" account. As with the above issue, I was always a bit uncomfortable with this in my years of use of Gnucash and I wanted to be sure that the Newcash user-interface didn't have the same problem.

In Figure 3.4, I've expanded the splits of the Mortgage Payment transaction. In Figures 3.5 and 3.6, you see the same transactions and splits, displayed by Newcash in its separate account and transaction registers, respectively. The latter eliminates the "overloading" of the account register with the job of displaying both splits and transactions. The hierarchy of financial objects, discussed in Section 4, tells us that Accounts contain Transactions and Transactions contain Splits. The Newcash user-interface uses the same organization.

---

[4]Note that Gnucash does not display the leading colons in account path names. Newcash does, for the same reason that Unix/Linux absolute file pathnames have leading forward-slashes. The leading character indicates that the path is from the root of the tree. Paths without the leading character are relative to an implied directory, or, in the case of Newcash, an account.

While some user-interface differences are already apparent, the balances are the same.

Figure 3.1: Gnucash: Checking Account, Traditional Labels

Figure 3.2: Gnucash: Checking Account, Explanatory Labels

If we now look at the register for the Liabilities:Home mortgage account

Figure 3.3: Gnucash: Checking Account, Paycheck Splits

Figure 3.4: Gnucash: Checking Account, Mortage Payment Splits

as displayed by the two systems (see Figures 3.7, 3.8 and 3.10), the difference
in balance calculation becomes apparent. In the Gnucash register, despite the
fact that a $100,000 credit is shown as a result of the "Purchase home" trans-
action, the balance after that transaction is positive $100,000. But that money

Figure 3.5: Newcash Account Register: Checking Account

	:Assets:Bank Accounts:Checking				
Date	Num	Description	R	Value	Balance
2013-05-15		Paycheck	☐	3000.00	3000.00
2013-06-01		Mortgage payment	☐	-1224.70	1775.30

2013-06-01 Mortgage payment				
Account	Memo	R	Value	Balance
:Expenses:House:Property taxes		☐	166.67	166.67
:Expenses:House:Mortgage interest		☐	391.17	557.84
:Assets:Bank Accounts:Checking		☐	-1224.70	-666.86
:Liabilities:Home mortgage		☐	666.86	0.00

Figure 3.6: Newcash Transaction Register: Mortgage payment Splits

flowed *away* from that account; it's a credit, as you can clearly see in Figure 3.8. Nonetheless, because of the sign-reversal rules we discussed earlier, Gnucash, following traditional double-entry practices by default, avoids the negative balance you would get by simply treating this arithmetically as a negative cashflow. Newcash does exactly that, resulting in a balance of -$100,000 for this account. You *can* get Gnucash to compute balances as Newcash does, by selecting a non-default preference that turns off the sign reversals. But if you do so with the explanatory column labels enabled, you will encounter a bug, present in the current release of Gnucash as of this writing (3.6); the program fails to reverse the labels in registers for account types that are subject to sign reversal by default: income, credit, liability, equity. See Figure 3.9, where you see that the value in the "Increase" column actually decreases the balance and vice-versa.

None of this dizzying array of column-labeling and balance-calculation options is present in Newcash; Newcash always presents values and computes bal-

Figure 3.7: Gnucash: Home Mortgage Account

Figure 3.8: Gnucash: Home Mortgage Account, Traditional Labels

ances in the one simple manner described earlier. Because I don't want to make an overly complex argument for simplicity, let us now end this discussion and move on to describing Newcash and its use in detail.

Figure 3.9: Gnucash: Home Mortgage Account, Sign Reversal Disabled

Figure 3.10: Newcash: Home Mortgage Account

Chapter 4

Financial Objects

In order to completely and accurately describe a user's finances, Newcash employs a set of financial objects in its database. Those objects can be seen in Figure 4.1, together with their inter-relationships. I describe these objects below, together with the data fields that comprise them in Newcash databases.

4.1   Books

A book points to the root of the tree of accounts that is the foundation of the Newcash accounting system. The tree is formed by child accounts pointing to parents, so the main function of the book is to allow us to find the root.

There are five types of accounts in the tree:

Assets What you own.

Liabilities What you owe.

Equity Net worth just prior to the start of record-keeping in the Newcash database.

Income Money you receive.

Expenses Money you spend.

The first two account types are so-called Balance Sheet items. A Balance Sheet is a snapshot, taken at a point in time, of the value of your Assets and Liabilities[1].

The Equity account is used for establishing opening balances for your Asset and Liability accounts when you begin the process of managing your finances

---

[1]The Balance Sheet that is included in the annual and quarterly reports of public companies typically is a snapshot of the state of assets and liabilities as of the close of business on the last day of the fiscal year or quarter.
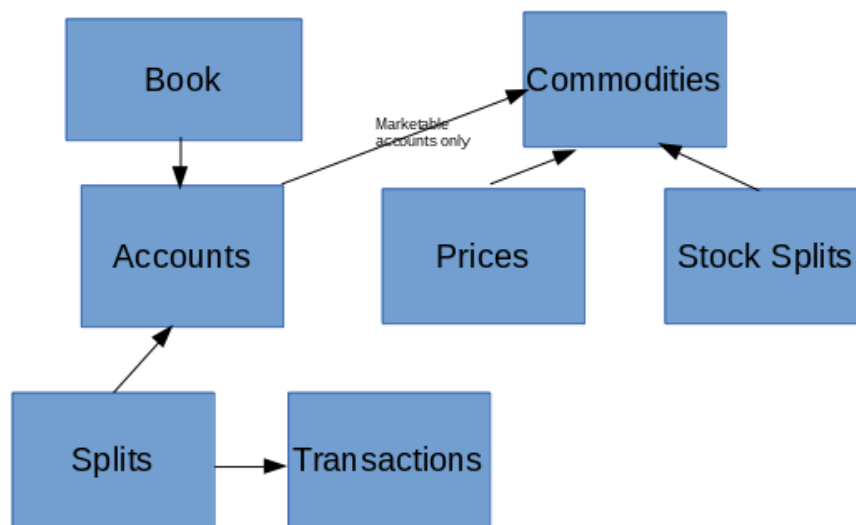
Figure 4.1: Financial Objects

with Newcash. After setting up your accounts in Newcash, one of your next steps will be to establish the current account balances of your pre-existing asset and liability accounts. Those balances are established by applying an opening-balance transaction to those accounts, causing a money flow either to or from those accounts. But remember, double-entry accounting requires that transactions consist of splits whose values sum to zero. So, for example, if your checking account has a balance of $855.47 at the time you begin setting up Newcash, you will want to create the checking account in Newcash and then create a transaction causing $855.47 to flow into that account. That flow has to come from somewhere and that somewhere is your Equity account.

Understand that while "Equity" and "Net Worth" are synonymous, the Equity account is *not* maintained by Newcash as your current Net Worth. That is calculated by the Newcash Report Generator from the current values of your assets and liabilities. The Equity account's sole reason for being is to provide a source/sink for opening balance transactions of asset and liability accounts and does not enter into the Net Worth calculation.

Note that the Equity account will have an opening balance of $0 when you begin. After entering the transactions to establish the opening balances of your Asset and Liability accounts, the balance of the Equity account will be the negative of your initial net worth. The sum of your Asset and Liability accounts at this point will be your initial net worth. That the balance of Equity is the negative of your initial net worth is completely inconsequential, as the Equity account is only important as the "other side" of opening-balance transactions, allowing you to properly initialize your asset and liability account balances. It can't be deleted, because it contains those opening balance transactions[2], but it can be ignored after you've done the initial setup of Newcash; you will have no further use for it. I would suggest making it a "hidden" account after you have finished setting up opening balances, to avoid cluttering the display of the account tree (Newcash, by default, does not display hidden accounts).

The last two accounts are so-called Income and Expense Statement items. An Income and Expense Statement[3] is a report on money flows to and from your income and expense accounts during a certain time period.

The account tree structure consists of a Root account at the top (so the tree is upside-down), with the five accounts described above as the Root's immediate descendents, its *child* accounts (these accounts are required in Newcash). In turn, those accounts can themselves have children. So you might have an account called "Bank Accounts" as a child of Assets. And "Bank Accounts" might itself have children – your checking account and savings account. Similarly, your credit card accounts and your mortgage would be children of Liabilities. Under

---

[2]You cannot delete any Newcash account that is referred to by at least one split.

[3]The Income and Expense Statement is traditionally referred to in corporate financial reports as an "Income Statement". But this is a misnomer, since those reports provide information about both income and expenses over a stated time period. We are already breaking with tradition in Newcash's handling of double-entry book-keeping, so I have taken the small step of changing the name of this report to reflect what it actually is. The Newcash Report Generator will produce both a Balance Sheet and an Income and Expense Statement for you.

Income, you might provide accounts for Salary, Dividends, Interest Received, Capital Gains, and so on. Expenses might have Food, Fuel, Utilities, Interest Paid, etc.

The Newcash account tree is analogous in many way to a Unix/Linux/ MacOS/Windows file-system, which is also a tree structure. Accordingly, the notation for the full path of an account from the root account of a Newcash account tree is identical to the notation for the full path of a file (or directory, which is a file) from the root directory of a Unix file-system, with two exceptions. I can describe one of the exceptions immediately: the nodes of a Newcash tree are separated with colons (":"), whereas the nodes of a Unix file-system tree are separated by forward-slashes ("/").

Let's look at an example of a file path. Suppose I have a file called Finances.newcash that is in a directory called Finances that is in my home directory, which is called dca, which is itself in a directory called home, which is a child of the root directory. That file is unambiguously described by the path `/home/dca/Finances/Finances.newcash`.

This is called an "absolute" path, which means that it shows you how to traverse the file-system tree from the root directory to the file in question. Note that the leading / denotes the root directory, not root/. This is because there is one and only one root directory in a file-system and that is where every absolute path originates. It is therefore unnecessary to say this explicitly every time you make use of an absolute path. This notation not only saves space and typing, but it makes possible "relative" paths, a topic I will not go into here but will simply mention that they do not begin with "/" and the absence or presence of a leading "/" enables the system to distinguish between absolute and relative paths.

Similarly, in Newcash, if I have an account called Joint Checking that is a child of Bank Accounts that is a child of Assets that is a child of Root, that account is unambiguously described by the path `:Assets:Bank Accounts:Joint Checking`.

Minor detail alert! One could argue that the leading colon is not necessary in Newcash account paths, since all account paths are absolute; there is no concept of relative account paths in Newcash, unlike Unix file paths. Gnucash makes this choice and displays all account paths without the leading colon. But there are situations in Gnucash where you type account paths and so this choice saves a bit of typing. This is not the case in Newcash; account paths are always selected via the graphical user interface, never typed. For this reason, I decided that it was better – less confusing – not to depart from the Unix convention in Newcash.

4.2   Accounts

Accounts are the objects that enable you keep track of what you own and owe, and money you receive and spend. All of the data fields described below can be set either when you create a new account, or by editing an existing account.

See Section 5.4.1 for detailed information on these operations.

Each account has a

Name The name you give an account is completely your choice, e.g., Cambridge Trust Joint Checking Account.

Parent The internal name of the account's parent[4]. For example, the Cambridge Trust Joint Checking Account might be the child of an account called Bank Accounts, so this field would point to that account.

Commodity If needed, the internal name of an associated commodity[5]. The only accounts that have valid internal commodity names are marketable asset accounts and income accounts whose income is derived from a commodity, such as an income account representing the dividends from a stock position.

Code This field is used to store account numbers. For most accounts, Newcash doesn't make use of this field, but there is one important exception: accounts for which you wish to use the Newcash Statement Importer, you *must* store the correct account number in this field. Examples of such accounts are liability accounts representing credit cards. See Section 5.6.4 for more information.

Flags Account flags are boolean values – they are either Set (True) or UnSet (False). These flags are (the first four are self-explanatory):

Descendents are Assets

Descendents are Liabilities

Descendents are Income

Descendents are Expenses

Descendents are marketable Certain assets are "marketable", in the sense that they can readily be bought and sold in public markets. I refer, of course, to stocks, bonds, options, futures contracts, and the like. Marketable assets need to be, and are, treated somewhat differently than other assets by Newcash, and so Newcash needs to know which assets are marketable. The "Descendents are marketable" flag provides this information. Assets that have an ancestor with the "Descendents are marketable" flag set, are, as the name suggests,

---

[4] "Internal names" of database objects in Newcash are randomly-generated 32-character strings, called GUIDs (Globally Unique IDentifiers), used throughout Newcash databases and the programs that manipulate them to name and refer to particular database objects. Their function is similar to that of the Vehicle Identification Number of your car (there is one and only one vehicle – yours – identified by your car's VIN). During normal operation of Newcash and its associated utility programs, you will not need to concern yourself with GUIDs. I mention them only to make you aware of their existence, which could be useful in an unusual situation, e.g., your database becomes corrupted due to a hardware problem.

[5] If not needed, this field contains a special value that, in effect, indicates that it is "not applicable".

regarded as marketable by Newcash. Accounts that are marketable must be associated with a marketable commodity (see ¡link needed¿ for more information about commodities). This association is established when the account is created, though it may be changed by editing the account.

Hidden Accounts with the "hidden" flag set are not displayed in the Book window by default (there is a Book window command to display hidden accounts). Designating an account "hidden" is useful when an account becomes dormant, e.g., a closed bank or investment account, but cannot be deleted because there are transactions that have splits that point to the account. Marking them "hidden" reduces clutter in the Account tree display in the Book window.

Placeholder Placeholder accounts are much like directories in file-systems; their purpose in life is to contain other accounts and group them. For example, you might have an asset account called "Bank Accounts" with all your bank accounts having this account as their parent. If you do this, then the Newcash Report Generator, in its Balance Sheet report, will display a value for "Bank Accounts" that is the aggregate value of all accounts below it.

Self and descendents are tax related Accounts that are marked "tax-related" are shown in italics in both the Balance Sheet and Income and Expense reports. If your accounts are properly organized and therefore tax-related accounts are grouped as children and deeper descendents of one account (usually a placeholder account), it will be sufficient to set this flag in just the one account at the top of your tax-related tree[6].

Descendents Generate Income From Commodities Some marketable commodities generate income, e.g., some stocks pay dividends, bonds pay interest. The return from such a commodity consists of both capital appreciation and income. In order for the Newcash Report Generator to properly compute the total return from a commodity in which you have a position, your income accounts need to be linked to that commodity, indicating the source of the income. The Report Generator will then be able to properly show the complete return from a position in such an income-generating commodity – capital appreciation plus income.

---

[6]Please note that this flag departs from the convention used in the other flags that propagate to descendents, in that it applies to the account in which it is set as well that account's descendents. This is simply so that the top-level account, e.g., "Dividends", will be italicized and therefore visually identifiable as tax-related, in the reports produced by the Newcash Report Generator(see Section 5.6.2). The value of that top-level account will sometimes be important at tax-preparation time, and it should be clear in the report that it is tax-related.

4.3 Transactions

Transactions describe the movement of money among potentially many accounts. The data contained by the transactions themselves is:

Number Despite its name, this is actually a field that can contain arbitrary text. For hand-written checks, you would probably enter the check number here. For payments made by electronic bill-paying, you would most likely enter the transaction identifier provided by your bank/bill-paying service.

Post date The date the transaction actually occurred.

Enter date The date the transaction was entered in Newcash. This date is set automatically for you and is not displayed in registers that show an account's transactions.

Description A description of the purpose of the transaction, e.g., "Mortgage payment".

4.4 Splits

Splits describe a money flow to or from exactly one account and are associated with exactly one transaction. They contain the following data:

Transaction The internal name of the transaction with which the split is associated. This field is not shown in the transaction registers that display the splits associated with a particular transaction. It is not necessary to do so because, in Newcash, you select a particular transaction and then request the register containing its splits. The window manager will display the description of the transaction in the title of the window in which the register is displayed. This will be discussed further later.

Account The internal name (not the account's name field that you chose) of the account with which the split is associated.

Memo Arbitrary text of your choosing, describing the split. It is similar to the Description field of the transaction, but that field is associated with the entire transaction; the Memo field is associated with a particular split.

Reconciliation state Represented internally as a Boolean value, true or false, indicating whether a split has been reconciled or not. In transaction registers, the reconciliation state field ("R") contains a check-box for setting this field.

Transfer Represented internally as a Boolean value, Set (True) or Unset (False), this field is only applicable to splits on marketable accounts. It indicates whether the flow represented by a split is a transfer of shares to a new account. This is important to the Report Generator, which provides capital gain information about your investments. To compute capital gains,

the Report Generator must compute the basis of a position. Without the Transfer indicator, transfers would look to the Report Generator like a position was closed in the old account and a new position opened in the new account (usually at a price of $0) and so the capital gains reported for transferred positions would use an incorrect value for the basis. In transaction registers, the transfer field ("T") contains a check-box for setting this field.

Value The value field of a split is the amount of money flowing to or from the account associated with the split. Positive values indicate inflows, negative values indicate outflows.

Quantity The quantity field applies only to splits associated with marketable accounts. It is the (signed) number of shares being transacted. Positive quantities indicate buying, negative quantities indicate selling. Note that the price/share is not stored explicitly in the Newcash database; it is computed from Value and Quantity:

$$Price\,per\,share = Value/Quantity \tag{4.1}$$

4.5 Commodities

Marketable asset accounts and income accounts that obtain their income from marketable securities must be associated with *Commodities*. Commodities are database objects representing marketable securities, such as stocks, bonds, options, or futures contracts. Commodity objects in the Newcash database carry the following data fields:

Symbol the security's ticker symbol, e.g., IBM.

Name The security's name, e.g., International Business Machines.

Cusip In order to accurately designate securities when trading or reporting (confirmations, brokerage statements), securities must have unique identifiers. Many people think a stock's ticker symbol serves that purpose, but it does not do so adequately. Symbols change and sometimes get re-used. Cusips are an attempt to provide more reliably unique security identifiers than do ticker symbols. From Wikipedia: "A Cusip is a 9-character alphanumeric code which identifies a North American financial security for the purposes of facilitating clearing and settlement of trades".

4.6 Prices

Price quotes are associated with exactly one commodity. Price quotes in a Newcash database carry the following data:

Commodity The internal name of the commodity to which the quote applies.

Timestamp The date and time of the quote.

Price The quoted price.

4.7   Stock Splits

Stock splits are associated with exactly one commodity. They are not to be confused with the Split objects described in Section 4.4; Stock Split objects serve an entirely different purpose.

If you think of a company as a pie, the company slices up that pie into "shares" and sells some of the shares in public markets. If, for example, a company slices the pie into 100,000 pieces, or shares[7], and you buy 100 shares, you now own $100/100000 = .001$ or .1% of the company. A stock split, can be thought of as putting the pie back together and re-slicing it, giving existing owners new shares in exchange for their old ones so that their fraction of ownership does not change.

For example, if the company splits its shares by a factor of 2 (a so-called "2-for-1" stock split), they have re-sliced the pie so there are now 200,000 shares and you will receive 200 of the new shares to replace your original 100. While the fraction of the company that you own has not changed ($200/200000 = .001$), the value of your position will also not change, because the price per share will halve the moment after the split.

Companies usually split shares in this way when their shares get very expensive, to enable someone who doesn't have a lot to invest to buy a small piece of the company. Similarly, they will sometimes use splits to *reduce* the number of shares, because very cheap (e.g., penny) stocks are avoided by some investors on the grounds that the low price is an indicator of too much risk.

Stock splits in a Newcash database carry the following data:

Commodity The internal name of the commodity to which the stock split applies.

Date The date of the stock split.

Factor The factor or multiple by which the stock has split.

---

[7]This company would be said to have 100,000 shares outstanding.

Chapter 5

Installing and Running Newcash

In the following discussion, I am assuming that you are familiar with Unix/Linux and that you are capable of running commands from a shell. I cannot provide a Unix/Linux primer in this document[1].

5.1 Building, Installing and Running Newcash From the Source Code

As mentioned at the beginning of this document, Newcash is written for Unix and Linux systems only. Most of the system was originally written in C, but I have recently re-written the C programs in Rust. Rust provides thread- and memory-safety that C does not and does so without sacrificing performance.

At the time of this writing, the C version of Newcash has been tested on a number of Linux distributions (Arch, Debian, Ubuntu, OpenSuse, and Slackware). It has also been tested and works properly on three of the four direct

---

[1]Using the term "Linux" to refer to the entire system is technically and politically incorrect. Linux is the name Linus Torvalds gave to the kernel that he wrote 25 years ago, the development of which he manages to this day. The issue is that the kernel is only one part, an important part to be sure, of the whole system. Another important part is the layer that sits immediate on top of the kernel and provides absolutely essential services without which the system would not be useable. That layer is provided by the GNU project, led by Richard Stallman, the founder of the free software movement. Stallman, quite rightly, has for years felt that the wide-spread custom of referring to the whole system as "Linux" was a slight to the important contribution of his project to that system. He advocates the term "GNU/Linux" instead. But there are other groups that provide essential components to the system, such as those who make "distributions", e.g., Debian, Ubuntu, Arch, Slackware, etc. Shouldn't their names be part of the system too? Having no desire to fight this battle here, I have decided just to use the most common term for the whole system, "Linux", in this document, with apologies to Richard Stallman, a person I greatly admire. I am a member of his Free Software Foundation.

descendents of Berkeley Unix: OpenBSD, FreeBSD and DragonFly[2][3].

The Rust version has been tested only on Debian Linux. Rust is generally available for Linux distributions and I would not expect any difficulty in running Newcash on distributions other than Debian (Alpine may be an exception, given its use of Busybox and an alternative C library, Musl). Rust is also available on FreeBSD and would expect the Rust version to work properly. Rust is also available as a package on DragonFlyBSD, but the version of the toolchain is usually behind the current version. For example, as this is written, the current version of Rust is 1.26. The version available as a binary package on Dragon-FlyBSD is 1.21. The Rust components of Newcash compile without errors or warnings with 1.26, but they do not compile with 1.21. Hopefully, DragonFly-BSD will become one of Rust's officially supported systems (it began life as a fork of FreeBSD, which is supported); it's a great OS. I am trying to do what I can to support that.

If you prefer to use the C version, it is available in a separate repository at github. Understand that I am no longer adding features to the C version (though, truth be told, I don't anticipate many additions to the Rust version either; after five years of managing my own finances with Newcash, it has long since reached the point where it has everything I need), though if a bug comes to my attention, I will fix it. Nonetheless, I recommend that you only use the C version if you are running an OS for which Rust is not available. I am using the Rust version for my own financial work and it will benefit from any issues I encounter. In addition, while the C version is generally reliable, there have been some unexplained crashes over the years that I have not been able to reproduce and, therefore, debug. The Rust language and its compiler, on the other hand, impose a level of rigor on the programmer that is far beyond that of C. Once you have satisfied the Rust compiler by writing code that is free of errors and warnings, you can be certain that your code is type-, memory-, and thread-safe. There will be no segmentation faults due to dangling references (references to freed memory) or dereferencing null pointers. There will be no race conditions in multi-threaded code. Any bugs will be in the logic or chosen algorithms. No such assurances can ever be made about code written in C.

The Newcash source code is made available via GitHub. To obtain a copy of the Newcash Git repository from that web-site, first insure that the Git application is installed on your system. Then, at a shell prompt, select the directory where you want the repository directory to reside, connect to it, and issue the command to clone the Newcash repository residing on GitHub. For example, if you want the Newcash repository directory to be located in a directory called `~/Software`, run the following commands with a shell:

```
cd ~/Software
git clone https://github.com/donaldcallen/newcash.git
```

---

[2]It is very likely that Newcash will also work correctly on NetBSD; I simply haven't tested it on that system.

[3]Given the availability of virtual machine software, Docker, and, in the case of Windows, Cygwin, I have no plans to port the system to Windows or the MacIntosh.

or

```
git clone https://github.com/donaldcallen/newcash_c.git
```

to install the C version.

The repository contains all of the Newcash source code, both the Rust and C versions. To build the system, you will have to insure that all libraries and packages on which it depends are present in your system.

The necessary components are:

- GNU Make, available as `make`. On BSD systems, you will likely need to install the gmake package, because the BSDs generally use the original version of make, not the GNU version on which Newcash depends. You will also likely need to set up a `~/bin/make` symbolic link to the gmake executable file, where-ever the package system placed it, e.g., `/usr/local/bin/gmake`. Your own bin directory should appear before the system directories in your `PATH`, otherwise `make` will refer to BSD make, which will not work correctly.

- If you are installing the Rust version, you will need the Rust compiler and toolchain. See https://www.rust-lang.org for installation instructions.

- A C compiler, available as `cc`. I have built Newcash with gcc and clang; either one is fine.

- The m4 macro processor.

- The GTK+3 library and header files.

- The sqlite3 library, header files and command line program.

- The sqlite3 Tcl package.

- Tcl

- pkgconf (for the C version)

- Texlive

Unfortunately, I cannot give you anything like complete instructions for how to obtain the above software for all Linux distributions. But I can give you examples.

For Ubuntu Linux, you will need to install the following packages:

- gcc

- libghc-zlib-dev

- libgtk-3-0

- libgtk-3-dev

- libsqlite3-0

- libsqlite3-dev

- libsqlite3-tcl

- m4

- make

- pkgconf

- sqlite

- tcl

- texlive-full

Having installed the necessary packages (and if you haven't gotten it quite right, the attempt to build the system will let you know what is missing, though the errors may not be particularly friendly).

Now try building Newcash. I assume in the following that you have installed my repository as described above.

```
cd ~/Software/newcash
make clean
make -j<number of cores in your system, +1>
```

If you encounter errors, it is almost certainly due to missing dependencies. You will have to diagnose the issue based on the received error messages and correct the problem.

If the system builds without error, congratulations! Before installing, you need to create `~/bin`, `~/bin/<OS name>`, and `~/lib` directories, where ¡OS name¿ is the same as the output of the `uname` command, e.g., 'Linux'. You must insure that both the `~/bin/<OS name>` and `~/bin` directories, in this order, appear in your PATH environment variable. With these preliminaries complete, install Newcash:

```
cd ~/Software/newcash
make install
```

Starting Newcash in point-and-click fashion depends on whether you are running Gnome, KDE, XFCE, or one of the other Linux desktop systems, or just a window manager. Usually, the simplest method is to click (or double-click) the Newcash database file in the file-system browser you are using. The browser might need to be told once which application to use to open the selected data file. Because there are so many different desktop/file-system browser choices available in Linux, it is beyond the scope of this document to provide details on all of them. I make an exception for Gnome, perhaps the most popular desktop. For Gnome users, I provide two files to assist you. After connecting to the `newcash` directory containing the Newcash repository, issue the following commands at a shell:

35

```
cd Gnome_registration_files
cp newcash.desktop /usr/share/applications
cp newcash-icon.xpm /usr/share/pixmaps
```

In Nautilus, the Gnome file-system browser, navigate to your Newcash database and right-click the file. A menu will appear. Click the "Open With Other Application..." item. The "Select Application" popup will appear. Click the "View All Applications" button at the bottom. Either scroll down to the "Newcash" item, or begin to type "Newcash" until it appears, highlighted. Click the "Select" button in the upper-right corner. If Newcash has been properly installed and your PATH environment variable is set correctly (includes

If you are using another desktop system, you will need to work this out for yourself, or simply start Newcash from a shell prompt, as shown at the beginning of this section.

5.2   Creating a New Newcash database

When you install Newcash, one of the scripts that gets installed is called

```
newcashCreateDatabase
```

As its name implies, this script will create a new Newcash database. This database will contain all the required accounts, but no transactions, splits, commodities, or prices; it is a blank slate. The script accepts a book name and a path to the new database file as command-line arguments, e.g.,

```
newcashCreateDatabase 'Joan and Don Allen Finances' \
~/Finances/Financial_management/Allen/Finances.newcash
```

5.3   Transferring Gnucash Data to Newcash

5.3.1   Storing Gnucash Data In A Sqlite3 Database

To transfer Gnucash data to Newcash, that data must reside in a Sqlite3 database. Gnucash provides a number of options for how it stores its data and Sqlite3 is one of them. If your Gnucash data is already stored in a Sqlite3 database, skip to Subsection 5.3.2. Otherwise, in Gnucash, with your financial data loaded, do a "Save As" and in the "Data Format" pulldown, choose Sqlite3; see Figure 5.1. Having produced a Sqlite3 version of your Gnucash data, convert that database to Newcash format as described in the next Subsection.

5.3.2   Converting a Gnucash Database to Newcash Format

Run the newcashConvertGnucashDatabase script that is installed when you do the make install discussed earlier:

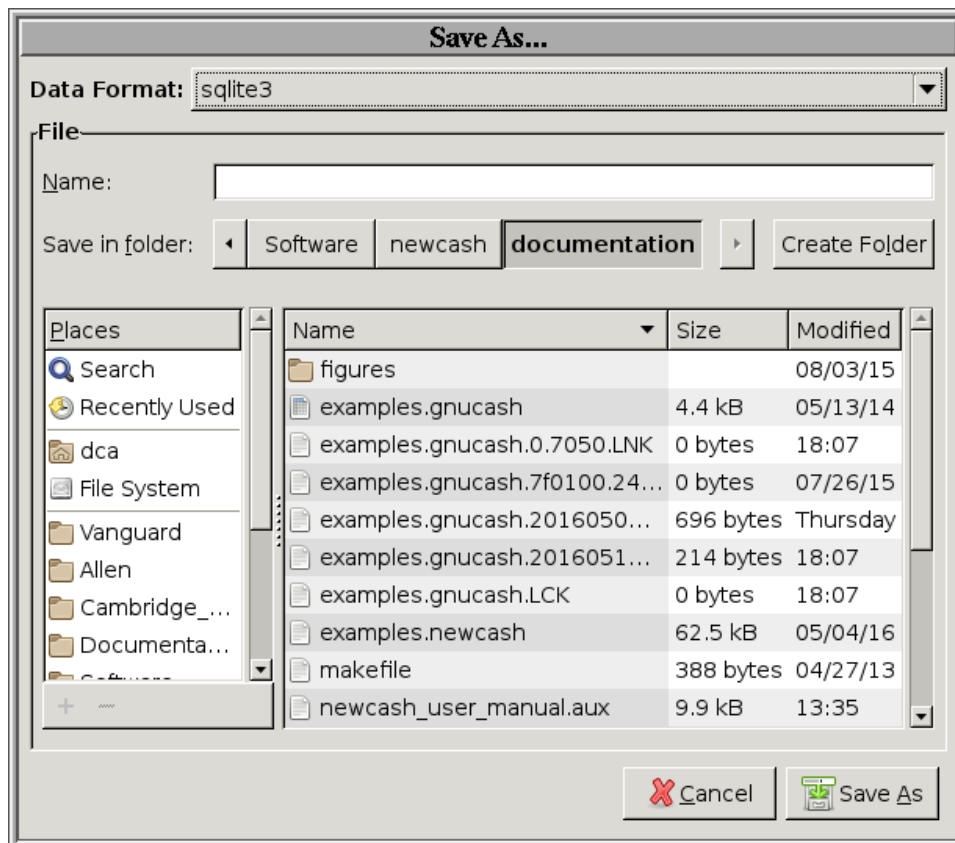Figure 5.1: Saving Gnucash Data In A Sqlite3 File

```
newcashConvertGnucashDatabase <name for your book> \
    <path to sqlite3 file written by Gnucash>
```

For example,

```
newcashConvertGnucashDatabase 'Joan and Don Allen Finances' \
    Finances.gnucash.sqlite3
```

The database will be converted to Newcash format in place; a separate transformed copy is *not* generated. So please be careful: if you have been storing your Gnucash data as a Sqlite3 database, do not do this conversion on your primary database file, because doing so will render your database unusable in Gnucash[4]. Instead, make a *copy* and run the conversion script on the copy. If you created a Sqlite3 version of your Gnucash data specially for your migration to Newcash (your primary Gnucash file is in another format, e.g., XML), then this caution does not apply to you.

Before proceeding with the description of the use of Newcash, some general comments about the user interface are in order:

- As mentioned earlier, Newcash provides no window-management, instead relying on the X window-manager you are using for that functionality. In particular, Newcash does not itself provide a way to close a window, because all window managers give you a way to do that. Nor does Newcash offer a way to quit the application; if you simply close the Book window with your window manager, Newcash will quit. You needn't worry about saving your work, because every change you make in Newcash is written to the database immediately. When you close the Book windBook, all of your changes will have already been written to the database and no last-minute requests to save your work will appear.

- To edit a field in a register, just click it or press the enter key ($\hookleftarrow$) to open it for editing. The field will be highlighted (as opposed to highlighting of the whole box in which the field resides). If you begin typing, you will obliterate whatever was present before, since the highlighting indicates that the whole field is selected, a standard user-interface convention since the days of the first MacIntosh. If you wish to change the present contents of the field without deleting all of it, either click in the specific place where you wish to edit, or navigate to it with $\rightarrow$ and/or $\leftarrow$. The Home and End keys can be useful in this context. The Home key will place the cursor to the left of the first character in the field, and the End key will place the cursor to the right of the last character.

---

[4]When making the transition from Gnucash to Newcash, it is advisable to use them both for awhile, until you become acclimated to Newcash. By this I mean that any changes, e.g., new transactions, should be made in both applications for a time.
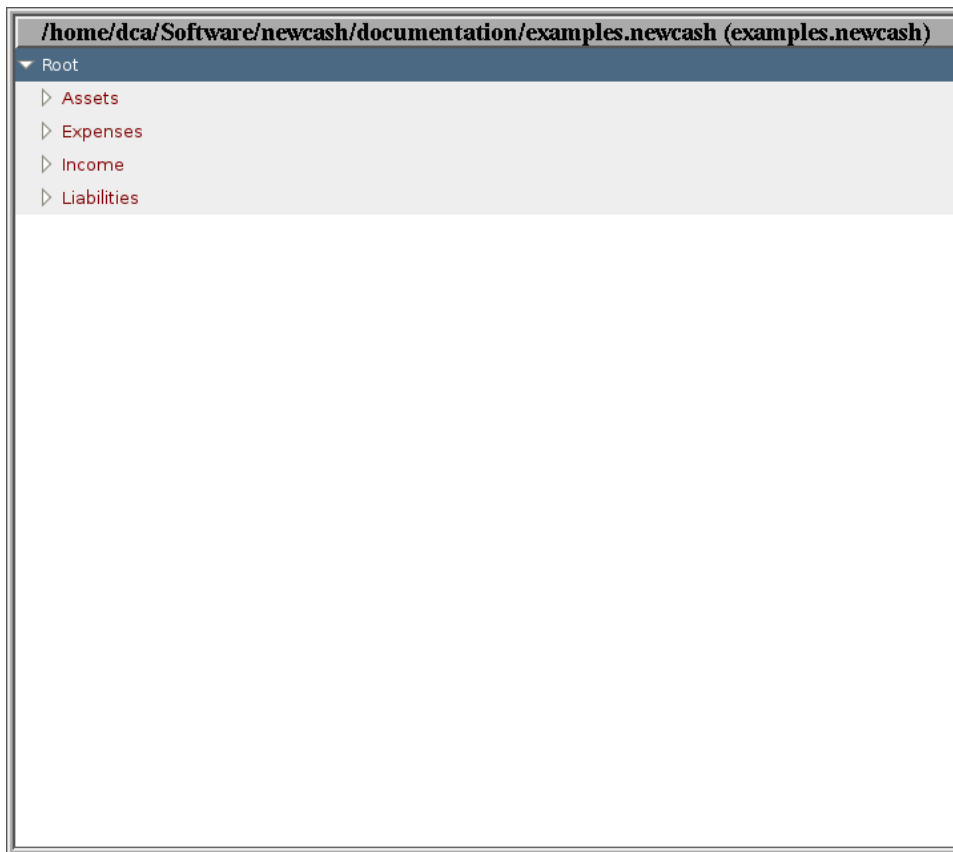
Figure 5.2: Book

5.4 Book Window

When you start Newcash, a window will appear displaying your book, the tree of your accounts. The title of the window will be whatever you named the book when you created the Newcash database. See Figure 5.2 for an example of the book window. You can left-click the little arrows to the left of the accounts to expand them (view their immediate descendents). This can also be accomplished from the keyboard by navigating to the desired account with the ↑ and ↓ keys and expanding by typing +. You can then collapse the account (hide the descendents) either again left-clicking the arrow to its left, or by typing −. Double-clicking a selected acccount, or typing enter (↩) will cause an account register to appear, displaying transactions that have at least one split pointing to the chosen account. I will discuss how to use that register in Section 5.5.1. But first, I will describe the book window's operations.
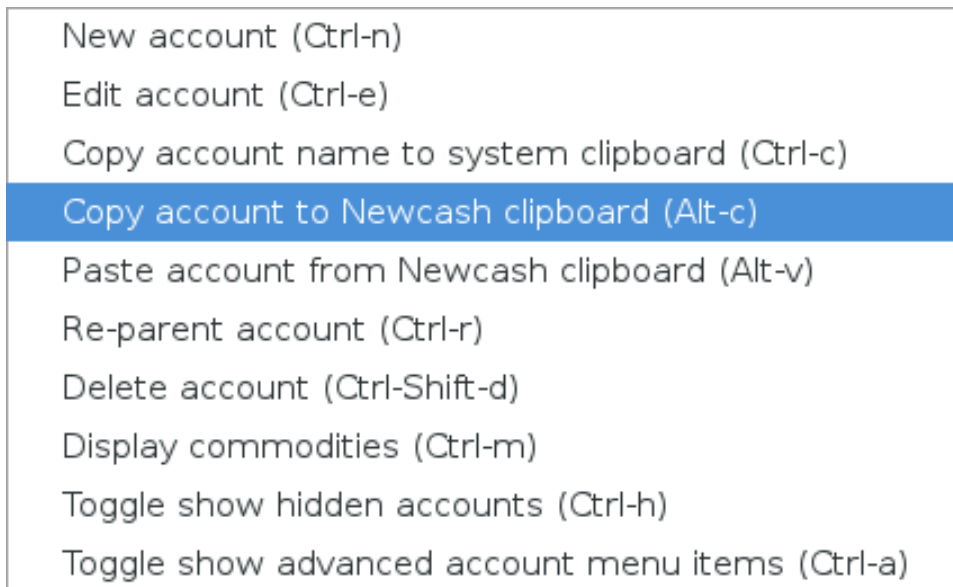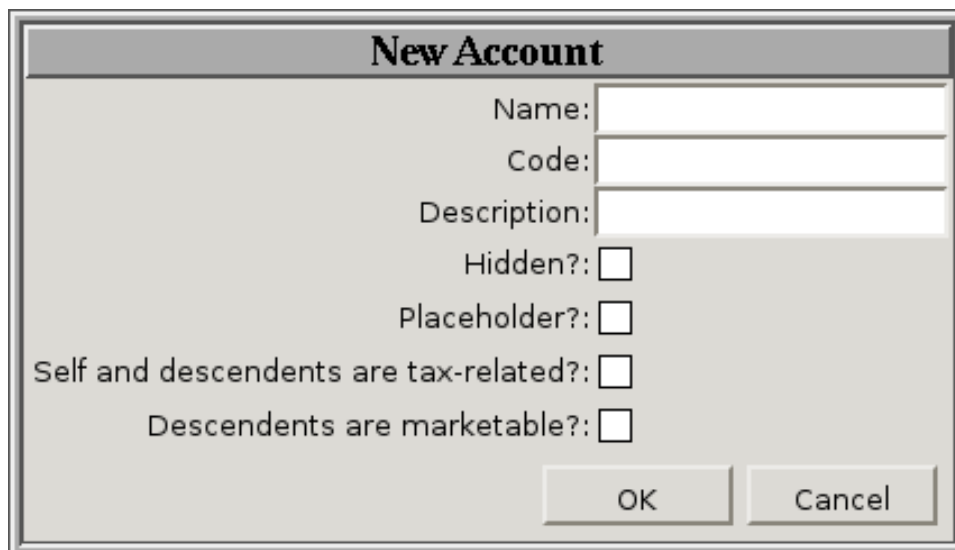
Figure 5.3: Book window Menu

Closing the book window with your window manager's "close" or "delete" operation will terminate Newcash. Consult your window manager's documentation for specific information on how to close windows.

5.4.1 Book Window Operations

The book window, like all Newcash windows, supports certain operations. These operations can be invoked either via a menu or from the keyboard. And where is that menu? You will notice immediately that there is no menubar on the main (or any other) Newcash window. Menubars occupy screen area even when they are not being used, which is most of the time. In Newcash, I've used pop-up menus exclusively, available, where appropriate, by right-clicking the window whose menu you wish to use. The menu entries vary, depending on the window. The functions performed by clicking the menu items can, in all cases, also be invoked from the keyboard. All descriptions of menu items in the subsequent discussion provide the applicable keyboard gesture in parentheses.

Right-clicking on the book window will pop up a menu (see Figure 5.3) that contains the items described below (right-clicking to obtain a window's menu is a convention supported by all Newcash windows that provide menus). Note that some of the menu items require that an account be selected prior to invoking them. This is done with a single left-click on the desired account, by use of the ↓ or ↑ keys, or by typing enough of the account name to specify it unambiguously (after doing so, you can indicate you are finished typing either by hitting `Esc`

Figure 5.4: New Account Dialog

or just by waiting a few seconds for the box in which your typing is echoed to disappear).
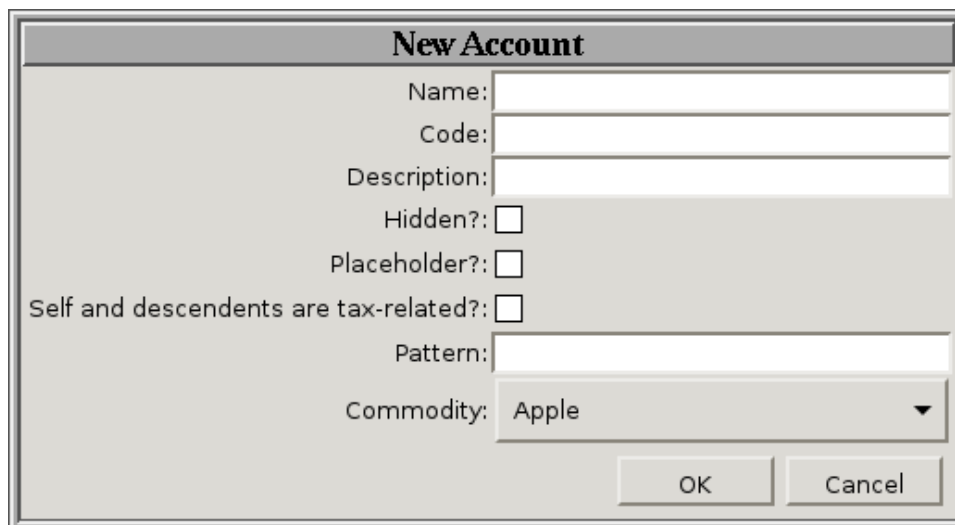
New Account (Ctrl-n) Before invoking this item to create a new account, you must first select the account that you want to be the new account's parent. Then invoke the New Account function. A dialog box will appear (see Figure 5.4) that allows you to set various parameters associated with the new account. They are as follows:

> **Name** The account name.

> **Code** If the new account represents an account held by an outside party such as a bank, brokerage, or credit-card issuer, this is the appropriate place to record the account number. If the new account has no association with an outside party and thus no account number, you may leave this field blank; it is purely for your own information and is not used by Newcash.

> **Description** A description of the account.

> **Hidden?** This is a checkbox that indicates whether the account is hidden or not. Hidden accounts are not shown in the account tree by Newcash by default. There is a menu item, described below, that allows you to show hidden acocunts. This capability is useful, for example, for hiding closed accounts so they don't create unnecessary clutter in the account tree.

Figure 5.5: New Marketable Account Dialog

Placeholder? This is a checkbox that indicates whether the account is a placeholder or not. A placeholder account's main reason for being is its children; no transactions are permitted for the placeholder itself. This allows you to group a set of accounts, the placeholder's children, that would otherwise be unrelated. This can be helpful for reporting purposes.

Descendents are tax-related? This is a checkbox that indicates whether the account's descendents are tax-related and will therefore be italicized in the Balance Sheet and Income and Expense Statement produced by the Newcash Report Generator (see Section 5.6.2).

Descendents are marketable? This field only appears in this dialog if the new account you are creating has an ancestor with the descendents-are-assets attribute (more simply, the account you are creating is an asset). Checking this box indicates that all descendents of this asset are marketable.

Descendents generate income from commodities? This field only appears in this dialog if the new account you are creating is an Income account (has an ancestor with the descendents-are-income-accounts property) and does not inherit the descendents-generate-imcome-from-commodities property from an ancestor. Checking this box means that when creating or editing descendents of the new account you are creating, you will be provided with a pattern and commodity pulldown with which to link the descendent to the commodity from which it obtains its income. This is usually used in the

42

case of Income accounts that represent dividend or interest income from equities, bonds, or mutual funds.

Pattern, Commodity These two fields appear in the New Account dialog only if the new account is marketable (an ancestor has the "descendents are marketable attribute") or inherits the "descendents generate income from commodities property". Accounts with either of these properties must have commodities associated with them. The commodity is the actual vehicle by which value flows in and out of accounts. Commodities are marketable securities, such as stocks, bonds, options, and futures contracts. The pattern field can be used to help narrow the displayed commodities to securities you are interested in in certain situations. If you provide a pattern, only those commodities whose names match the pattern will be displayed. The "%" character serves as a "wildcard" character[5] and will match any sequence of characters, including the null sequence (no characters). You may use the wildcard character as often as necessary in a pattern. For example, the pattern

`%Apple%`

will match any commodity whose name begins with any character sequence, including no characters at all, followed by "Apple", followed by any other character sequence. You might be thinking "There's only one Apple stock, why is this useful?". Suppose you trade Apple options and do so over a long period of time. You will accumulate a large number of commodities containing the string "Apple", such as "Apple 2013-05-10 460 calls", "Apple 2012-11-18 550 puts", etc. When creating a new account for such a commodity, you will find that the pattern and wild-card character make the commodity easier to locate in the pulldown.

Edit Account (Ctrl-e) Before invoking this item to edit an existing account, you must first select the account that you wish to edit. The same dialog box will appear as when creating a new account. See the above description of the new account dialog for details.

Copy account name to system clipboard(Ctrl-c) Before invoking this command, you must first select the account that you wish to copy. This command does what its name implies – it copies the name of the selected account to the system clipboard. You can then paste the name anywhere the pasting operation is available, e.g., the description field of a transaction, a spreadsheet cell.

Copy account to Newcash clipboard (Alt-c) Before invoking this item, you must first select the account that you wish to copy. The Newcash clip-

---

[5]For those who have written relational database queries in the SQL language, you will recognize the "%" character as the wildcard character in "LIKE" expressions that may be used in "WHERE" clauses. This is exactly how the pattern field is used internally in Newcash.

board is, as its name implies, internal to Newcash and can be used only for operations implemented by Newcash. It is distinct from the system clipboard. But like the system clipboard, the Newcash clipboard can hold only one account at a time, so if you invoke this command multiple times on different accounts, only the last account guid will be present in the clipboard. This command allows you to make a copy of an account, using the pasting operation described below. It is also useful when creating new transactions or editing existing ones. Specifically, when you display the splits for a transaction in a Transaction Register, you can paste the account held in the Newcash clipboard into a split's account field. This is discussed in greater detail in Section 5.5.2.

Paste account from Newcash clipboard (Alt-v) Before invoking this command, you must first select the account that you wish to be the parent of the new account created by the pasting operation. This command will create a new account having identical attributes to the account you copied to the Newcash clipboard. It will be a child of the account you selected prior the invoking this command.

Re-parent account (Ctrl-r) Before invoking this command, you must use "Copy account to Newcash clipboard" to copy the account you wish to be the new parent account. You must then select the account you are re-parenting. This command regenerates the display of the account tree in the Book window after performing the re-parenting operation.

Delete Account (Ctrl-shift-d) Before invoking this item to delete an existing account, you must first select the account that you wish to delete. Newcash will not allow you to delete an account for which there are existing transactions, or an account that has child accounts. If you are attempting to delete such an account and the deletion fails (Newcash will inform you of that with a warning message), perhaps you would be better served by designating the account as "hidden".

Display Commodities (Ctrl-m) This command will display a new window containing a register of all of the commodities you have defined. An example of this register is shown in Figure 5.10.

Toggle show hidden accounts (Ctrl-h) Accounts may be designated as "hidden", using the account-editing dialog described previously (you can also define a new account as hidden, though this is likely to be a rare occurrence). By default, Newcash does not display hidden accounts in the account tree in the book window. Invoking this command the first time in a Newcash session will cause hidden accounts to be displayed (the book window title will include a parenthetical notice that that is the case). Invoking it again will return Newcash to its default, hiding accounts marked "hidden". An operation like this that turns something on and then off upon repeated invocations is called a "toggle".

5.5 Registers

Before proceeding with the descriptions of Newcash's account and transaction registers, it is necessary to briefly discuss the editing operations common to both of them. Most, but not all, Newcash register cells are editable. You must begin editing by "opening" the cell, either clicking it once , or by pressing the `enter` key with the cell selected (the selected cell will have a dotted line around its contents).

If you prefer using the keyboard for most user-interface operations, as I do (this is common among computer professionals, in my experience, because it is more efficient for most operations), you can navigate from row to row with the ↑ and ↓ keys, and from cell to cell within a row with the → and ← keys.

Once a cell has been opened, you can change its contents in normal, text-editing fashion. In addition to use of the mouse, you can move the cursor within an open cell with the → and ← keys, as well as the `Home` key, which will take you to the beginning of the cell's contents, and the `End` key, which will take you to the end.

Notice that when you first open the cell, the entire contents is highlighted, so if you just begin typing, you will obliterate what is already there. To do more surgical editing, you must position the cursor either with the mouse or the keyboard. Once you have finished editing, either click anywhere in the Register other than the cell you are editing, or just press the `enter` key, and you will have recorded (committed) the new contents. If you change your mind and wish to abort your editing prior to committing your changes, press the `Esc` key and the original contents of the cell will be restored. But once you have committed a change to a cell, it has been written to your Newcash database. If you now wish to undo your changes (or to edit them further), you will have to re-open the cell and edit it however you wish.

Copy-paste to/from the system clipboard works with all editable fields. To copy the entire contents of an editable cell, open the cell and press `Ctrl-c`. To cut, press `Ctrl-x`. To paste, press `Ctrl-v`. You can paste into a specific location in the contents by positioning the cursor to where you wish to insert the pasted characters. The cut and paste operations make changes to an open editable cell, just as if you had manually made those changes. Like manual changes, they must be committed as described earlier if you wish to retain them.

5.5.1 Account Register

As mentioned earlier, double-clicking an account in the Book window will create an Account Register window, displaying all transactions related to the chosen account, sorted in ascending order of the date on which the transaction occurred. By "related" transactions, I mean those transactions that contain at least one split whose account is the chosen account.

Four of the columns, or fields, displayed in the account register are editable; you may change them directly in the Register (see Section 5.5 for instructions on editing Newcash cells). Those fields are Date, Num, Description, and R

(reconcilation state). The first three are simple text fields, whereas the R field is a checkbox and may be changed by clicking it, or pressing the `space` bar with the field selected.

:Assets:Investments:Equities and derivatives:Taxable:Vanguard:Tesla Motors							
Date	Num	Description	R	Shares	Price	Value	Share balance
2013-05-03		Buy Tesla	☐	300.0000	54.5500	16365.00	300.0000

Figure 5.6: Account Register, Marketable Account

Note that account registers for marketable asset accounts are a bit different than those for all other accounts. See Figure 3.5 for an example of an account register for a non-marketable account. Figure 5.6 is an account register for a marketable asset account.

Account Register Fields

In this and subsequent sections on the fields displayed by Newcash's various registers, there will be some duplication of material that already appeared in Chapter 4, which describes the objects in Newcash databases. The registers are, of course, derived from those database objects, but they don't display those objects verbatim in all cases. And, in some cases, there are fields in the registers that are computed from the contents of the database, e.g., prices in splits on marketable accounts and the running balances in account registers. Also, the registers are interactive; they are intended to allow you to edit the contents of the database in a controlled way, not just view it. Therefore, the material here describes their user interfaces.

I will now describe the account register columns/fields in detail:

Date field The Date field is the date on which the transaction with which it is associated occurred. As described in the previous paragraph, you can open a Date cell and enter a new date manually, in the form YYYY-MM-DD. But the Date column also supports the entry of special characters that make date entry easier (to avoid repetition, note that in the following, I am assuming that you have opened a date cell and the character or characters described below are the next things you type, replacing whatever was in the cell previously):

46

t will enter the current date into the cell.

= will increment the date currently in the cell by one day.

+ will increment the date currently in the cell by seven days.

- will decrement the date currently in the cell by one day.

_ will decrement the date currently in the cell by seven days.

m will enter the first date of month of the date currently in the cell. For example, if the cell contains 2014-03-27 and `m` is entered in that cell, the cell will be changed to 2014-03-01.

h will enter the last date of month of the date currently in the cell. For example, if the cell contains 2014-03-27 and `h` is entered in that cell, the cell will be changed to 2014-03-31.

Any (optionally signed) integer will add the specified number of days to the existing date in the cell. For example, if the cell contains 2014-03-04 and `4` is entered, the cell will be changed to 2014-03-08. If `-9` is entered, the cell will be changed to 2014-02-23.

All date arithmetic calculations are done with full awareness of calendar special cases, such as leap years. For example, if a date cell contained 2004-03-04 and you entered `-9` in the cell, the resulting date would be 2004-02-24, since 2004 was a leap year.

If you enter any text into a Date cell other than the special entries described above, Newcash will expect the entry to be a date in the form YYYY-MM-DD. If your entry is not in that form, you will receive an error message and the cell will not be changed.

Date fields may also be edited by use of a calendar. You can obtain a calendar in order to change the date in a specific transaction by first selecting the transaction either by left-clicking it or moving to it with the ↑ or ↓ keys. Once the row is selected, you can pop up a calendar either via a menu item or by issuing a command from the keyboard, both methods to be described shortly.

Num and Description fields Both of these fields are simple text fields and you can enter any text you wish. When I pay bills using my bank's Web-based bill-payment service, I copy the confirmation number it provides and paste it into the Num field when entering the transaction. The Num field is also the place to record the numbers of hand-written checks.

R The R field is a simple checkbox, indicating whether the transaction has been reconciled or not. Change it either by clicking or by selecting and pressing the `space` bar. This field appears in the account register as a convenience, because the reconciliation state is actually a property of splits, not transactions. When you alter the state of the R field of a transaction, you are actually changing the reconciliation state of splits associated with that transaction. The affected splits (usually only one) will

be those pointing to the account whose transactions are displayed by the account register. For example, if you double-clicked `Checking` in the book window, an account register would appear, containing all the transactions having at least one split associated with (pointing to) `Checking`. If you mark a transaction reconciled by clicking the R field, you have really marked the split belonging to that transaction and pointing to `Checking` as reconciled. You could accomplish the same thing by displaying the Transaction Register for the transaction you wish to mark reconciled, and clicking the R field for the appropriate split there, but that involves more user-interface gestures and would be very cumbersome when reconciling a statement. This is precisely the reason why this field appears in the account register.

Marketable Accounts: Shares, Price, Value and Share Balance Fields These fields are not editable in the account register. As described below, they are derived from data actually stored in splits, not transactions. Editing operations on these fields must be done from Transaction Registers, described later. Only account registers of asset accounts that are designated marketable (by virtue of having an ancestor with the descendents-are-marketable attribute) have Shares, Price, Value and Share Balance columns. These columns are all actually properties of the transaction's split that points to the account displayed by the account register, except for Price, which is not explicitly stored, but rather is computed using the simple relation $Price = Value/Shares$.

The Share Balance is a running balance of the number of shares in the account's positions.

Non-marketable Accounts: Value and Balance Fields The account registers of all account types other than assets and asset accounts that are not marketable will have only Value and Balance columns. The Value column contains the value of each transaction's splits that points to the account displayed by the account register. The Balance column is a running balance of those values, computed by simple addition of the signed values, as discussed at length earlier.

Account Register Operations

The account register window supports a number of operations that can be invoked either via menu or from the keyboard. Per the Newcash convention mentioned in the previous section, the menu will appear in response to a right click in an account register window. The operations are as follows:

New transaction (Ctrl-n) creates a new transaction, not surprisingly. The transaction will have the same date as the latest date of the existing transactions in the account register. It will consist of two splits, one pointing at the account associated with the account register, the other pointing at the `Expenses:Unspecified` account.
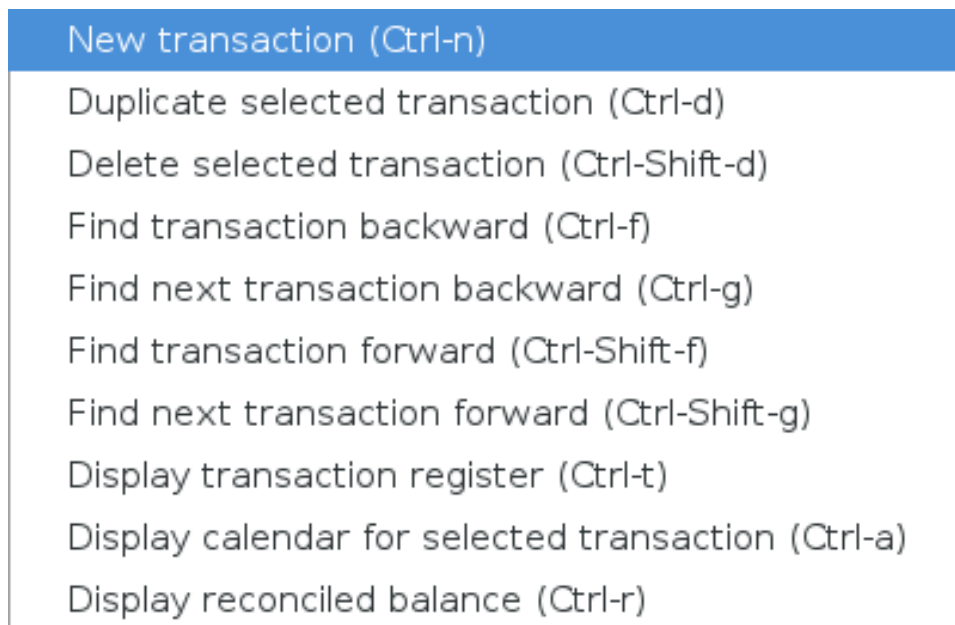
New transaction (Ctrl-n)
Duplicate selected transaction (Ctrl-d)
Delete selected transaction (Ctrl-Shift-d)
Find transaction backward (Ctrl-f)
Find next transaction backward (Ctrl-g)
Find transaction forward (Ctrl-Shift-f)
Find next transaction forward (Ctrl-Shift-g)
Display transaction register (Ctrl-t)
Display calendar for selected transaction (Ctrl-a)
Display reconciled balance (Ctrl-r)

Figure 5.7: Account Register Menu

Duplicate selected transaction (Ctrl-d) is very useful in conjunction with
the Find command, described below. The selected transaction (and its
associated splits) is duplicated, giving it the same date as the latest date
of the existing transactions in the account register. It is common to have
recurring transactions that are largely or completely identical, but if they
vary, the differences are not predictable. Your electric bill is an example
of this. The amount of each electric bill you receive varies, but you will
usually pay it from the same account, perhaps your checking account,
describe it the same way, and attribute it to the same Expense account,
e.g., `Expenses:Home:Utilities:Electricity`. Each time you pay that
bill, you don't want to have to create a new transaction and re-enter all
the constant information. A much simpler method is to use the Find (and
perhaps Find Next) command to locate the previous electric bill payment,
duplicate it, adjust the amount and date of the duplicate transaction and
you are done.

Delete selected transaction (Ctrl-Shift-d) deletes the selected transaction
and its associated splits.

Find transaction backward (Ctrl-f) uses a regular expression that you en-
ter in a dialog to search backward in time through an account register's
transactions, starting with the latest one. For a brief tutorial on reg-

49

ular expressions, see Section A. You can cause the Find command to match on any of the account register's fields. The Description field is the default. As mentioned just previously, the Find commands are very useful in conjunction with the *Duplicate selected transaction* command, to avoid unnecessary work when entering transactions that are almost or completely identical to previous transactions.

Find next transaction backward (Ctrl-g) will repeat a previous Find (which may have been either forward or backward), beginning with the selected transaction. The search is backwards in time. Attempting this command without having done a previous Find is an error.

Find transaction forward (Ctrl-Shift-f) is identical to its backward counterpart, just described, with the sole exception that the search is forward in time.

Find next transaction forward (Ctrl-Shift-g) is identical to its backward counterpart, just described, with the sole exception that the search is forward in time.

Display transaction register (Ctrl-t) displays a transaction register for the selected transaction. Transaction registers display a transaction's splits and are described more fully in Section 5.5.2. As a convenience, in addition to this menu item and keyboard command (Ctrl-t), a transaction for the selected transaction can be displayed by a single click of the middle mouse button.

Display calendar for selected transaction (Ctrl-a) displays a calendar, allowing you to alter the date of the selected transaction.

Aside from the obvious use of the mouse to select year, month, and day-of-month, the calendar supports useful operations from the keyboard:

Arrow keys The Arrow keys allow you move about within the days of the month.

Space The space bar selects the date you have navigated to with the arrow keys.

Ctrl+→ moves the calendar to the next month.

Ctrl+← moves the calendar to the previous month.

Ctrl+↓ moves the calendar to the next year.

Ctrl+↑ moves the calendar to the previous year.

Tab moves among the objects in the calendar window, including the buttons.

Display reconciled balance (Ctrl-r) produces a dialog displaying the balance of the reconciled transactions in the account register. This is useful after marking transactions reconciled from statement; it should match the

statement's balance. If not, you have a reconciliation error that you will
have to resolve with some detective work. Suggestions:

- Compute the difference between the statement balance and the recon-
  ciled balance reported by Newcash and do a "Find" for a transaction
  having that amount. If the transaction is supposed to be marked
  reconciled, be sure it is.

- Compute the difference between the statement balance and the rec-
  onciled balance reported by Newcash and do a "Find" for a trans-
  action having *twice* that amount. You may have entered the correct
  amount for the transaction, but got the sign wrong. If you did, your
  reconciled balance will be incorrect by twice the absolute value of the
  transaction value.

- If you know that the previous month reconciled correctly, then the
  error must be in the current statement period. Go through the state-
  ment period's transactions again, making sure that the amounts en-
  tered in Newcash match the amounts in the statement.

Account registers are closed by use of your window manager's "close" or
"delete" operation. Please consult your desktop system or window manager
documentation if you don't know how to close a window.

5.5.2   Transaction Register

Transaction registers display the splits associated with a particular transaction.
As described above, you may view a transaction register by first selecting the
desired transaction in an account register and then

- right-clicking to pop up an account register menu and clicking the *Display
  transaction register* item, or

- typing Ctrl-t, or

- clicking once with the middle mouse button.

See Figure 3.6 for an example of a transaction register. The transaction
register displayed from account registers for marketable Asset accounts is a bit
different from the transaction register for all other transactions. See Figure 3.6
for an example of an transaction register for a transaction from a non-marketable
account. Figure 5.8 is an transaction register for a marketable transaction.

Transaction Register Fields

The fields/columns of the transaction register are now described.

Account As mentioned previously, each row of a transaction register is a split.
Splits specify the movement of a certain amount of money to or from an
account. This field specifies that account. The field, as displayed in the

2013-05-03 Buy Tesla							
Account	Memo	R	T	Shares	Price	Value	Balance
:Assets:Investments:Equities and derivatives:Taxable:Vanguard:Tesla Motors		☐	☐	300.0000	54.5500	16365.00	16365.00
:Assets:Investments:Cash and cash equivalents:Taxable:Vanguard		☐	☐			-16370.00	-5.00
:Expenses:Investments:Commissions		☐	☐			5.00	0.00

Figure 5.8: Transaction Register: Marketable Account

transaction register, contains the full path to the split's account, but you do not change a split's account by editing this text. Instead, you change a split's account by pasting an account previously copied from either the Book window's account tree, or from another split, perhaps belonging to a different transaction. This will be discussed in detail in Section 5.5.2.

Memo The memo field is arbitrary text that you may enter. It plays a similar role as the Description field in account registers, with the difference that a Description applies to an entire transaction, whereas a Memo applies to a particular split. This field is editable.

R The R field is the reconciliation state of a split. It was discussed earlier in Section 5.5.1. Edit by either clicking the checkbox or selecting it and pressing the `space` bar.

Fields Specific to Transactions Involving Marketable Accounts The following fields are only displayed by transaction registers that represent transactions from an account register associated with marketable asset accounts.

T (Transfer) This field indicates whether the money flow represented by a split is a transfer of marketable assets from one account to another. This is important in report generation, particularly the report on the status of open investment positions. If you move shares (without liquidating the position) from one account to another without setting the T flag, the move will look to the report generator like a closing transaction in the original account followed by an opening transaction in the new account. The report generator will therefore calculate your current unrealized capital gain using the price of the apparent opening transaction. If, on the other hand, you set the T flag on the split showing the shares leaving the original account and on the split representing the arrival of the shares in the new account, the report generator will ignore both transactions and the basis used will be the original (correct) basis. Edit by either clicking the checkbox or selecting it and pressing the `space` bar.

Shares The number of shares transacted resulting in the money flow represented by the split. Positive share amounts represent purchases of shares, negative share amounts indicate sales. This field is editable.

52

Price The price/share of the shares transacted. This field is not explicitly stored in the database. It is computed from the simple relation $Price = Value/Shares$. Note that the convention used for the sign of the Shares field results in the Price always being a positive quantity, as it should be. This is due to the fact that the sign of the Value field for a split having a non-zero Shares will always be the same as the sign of the Shares field and thus the ratio $Value/Shares$ will always be positive. Why is the sign always the same? Because such a split must point to a marketable account (e.g., a stock or mutual fund) and when buying, the money flow in the transaction will be from the split representing the cash account to the split representing the marketable account (which represents the security you are buying) and therefore the latter must be positive. From the above description of the Shares field, we see that that field is positive for purchases and so the price in this case is the ratio of two positive numbers and therefore positive. When selling, the money flow will be *from* the marketable account to your cash account and therefore the value field of the split representing the security will be negative in this case. But the convention for the Shares field tells us that that field will also be negative, and so the price will again be positive (the ratio of two negative numbers is positive; remember that use of Newcash requires the ability to remember some of your elementary- or high-school math!).

This field is editable.

Value The signed amount of money flowing to (positive) or from (negative) the account named in the split's Account field. This field is editable. In addition to entering signed numbers in this field, you may also enter arithmetic expressions, e.g., $3056.12 - 9.17$.

Balance This is the running balance of the value fields of the splits in the transaction register. It is important to understand that the final value of the balance field in a transaction register must be zero. This is a fundamental principle of double-entry accounting[6]. Please note that if you attempt to close a transaction register that does not obey this principle and is not "balanced" (does not have a final balance of zero), you will receive an error message from Newcash and the transaction register will remain open, allowing you to correct your error. Also, when you attempt to quit Newcash by closing the book window, Newcash checks whether there are any open transaction registers that are unbalanced. If so, it will

---

[6]For those who have taken courses in the analysis of electronic circuits, this is analogous to Kirchoff's Current Law, which states that the algebraic sum of the currents flowing to/from a single node must be zero. It makes perfect sense – charge does not "pile up" in circuit nodes, which are simply places where multiple wires meet. And money doesn't come from nowhere, nor does it magically disappear; hence the double-entry principle. These are both examples of the conservation laws that those of us who have studied science and engineering have encountered repeatedly.
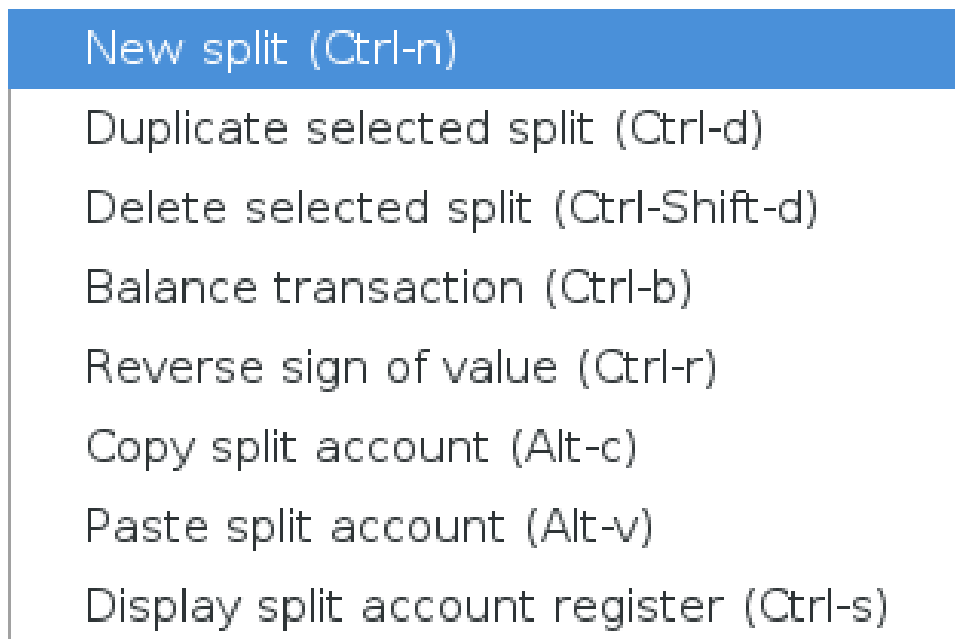
Figure 5.9: Transaction Register Menu

issue an error message informing you of the problem and will not quit, so that you can correct your error.

Transaction Register Operations

As with account registers, transaction registers support a set of operations, which may be invoked either from a menu, viewable by right-clicking a transaction register, or from the keyboard (the particular keyboard gestures are given with the description of each operation below).

New split (Ctrl-n) creates a new split pointing to the `Expenses:Unspecified` account and with null or zero values for the other editable fields.

Duplicate selected split (Ctrl-d) requires that you first select one of the splits in the transaction register. Issuing this command will then duplicate that split. Failure to first perform the selection will result in an error.

Delete selected split (Ctrl-Shift-d) requires that you first select one of the splits in the transaction register. Issuing this command will then delete that split. Failure to first perform the selection will result in an error.

Balance transaction (Ctrl-b) requires that you first select one of the splits in the transaction register. As explained earlier, a fundamental principle

54

of double-entry accounting requires that the values of a transaction's splits sum to zero. When you enter a transaction with two splits and alter the value of one of the splits, Newcash will automatically change the value of the other split to balance the transaction (unless the other split points to a marketable account, where changing the value without an adjustment of the quantity could produce an unwanted result; it could even be a hidden result, if the transaction register came from an account register of a non-marketable account and thus the transaction register lacks quantity and price columns; this situation is best handled by you, the user). But when more than two splits are involved, Newcash cannot do this for you, since it has no way of knowing how to allocate the money flows. You must do this manually in this case. This command helps with this task by changing the value of the selected transaction so that the final balance of the transaction will be zero.

Reverse sign of value (Ctrl-r) requires that you first select one of the splits in the transaction register. As we have already discussed, the sign of a split's value has great significance, indicating the direction of money flow. You will find that occasionally you will enter the absolute value[7] of an amount correctly but get the sign wrong. This command allows you to easily correct this error.

Copy split account (Alt-c) requires that you first select one of the splits in the transaction register. This command copies the selected split's account to the Newcash account clipboard. You can later paste this account into another split.

Paste split account (Alt-v) requires that you first select one of the splits in the transaction register. This command pastes the contents of Newcash's account clipboard to the account field of the selected split. An account can be placed in the clipboard either by using the command described just previously, or by using the book window's

```
Copy account to Newcash clipboard
```

command, described in Section 5.4.1.

Display split account register (Ctrl-s) requires that you first select one of the splits in the transaction register. This command displays an account register for the selected split's account.

5.5.3 Commodities Register

Commodities Register Fields

The columns of this window are:

---

[7]Formally, $abs(x) = x, x \geq 0$ and $abs(x) = -x, x < 0$. In plain English, the absolute value of a positive or zero quantity is just that quantity. The absolute value of a negative quantity is that quantity with its sign reversed. $abs(5) = 5 = abs(-5)$.

Commodities		
Symbol	Name	CUSIP
AAPL	Apple	037833100
IBM	International Business Machines	459200101
TSLA	Tesla Motors	88160R101

Figure 5.10: Commodities Register

Symbol The commodity's ticker symbol. This field is editable.

Name The name of the commodity, e.g., "International Business Machines", as opposed to its symbol, e.g., "IBM". This field is editable.

CUSIP From Wikipedia: "A CUSIP is a 9-character alphanumeric code which identifies a North American financial security for the purposes of facilitating clearing and settlement of trades." CUSIPs are an attempt to provide a better unique identifier for marketable securities than their ticker symbols, which can change and be recycled (assigned to different companies at different times). This field is editable.

Commodities Register Operations

Right-clicking on the commodities register will pop up a menu (see Figure 5.11) that contains the items described below. Note that some of the menu items require that an account be selected prior to invoking them. This is done with a single left-click on the desired account, by use of the ↓ or ↑ keys, or by typing enough of the account name to specify it unambiguously (after doing so, you can indicate you are finished typing either by hitting `Esc` or just by waiting a few seconds for the box in which your typing is echoed to disappear).

New Commodity (Ctrl-n) A new row is created at the top of the commodities window and is selected. Edit the fields to describe a new commodity.

Duplicate Selected Commodity (Ctrl-d) You must select a commodity before invoking this item. The selected commodity will be duplicated, with the string "(Copy)" appended to the name. This is useful if you trade a number of instruments, such as options or futures, that have similar names.

Delete Selected Commodity (Ctrl-shift-d) You must select a commodity before invoking this item. The selected commodity will be deleted.
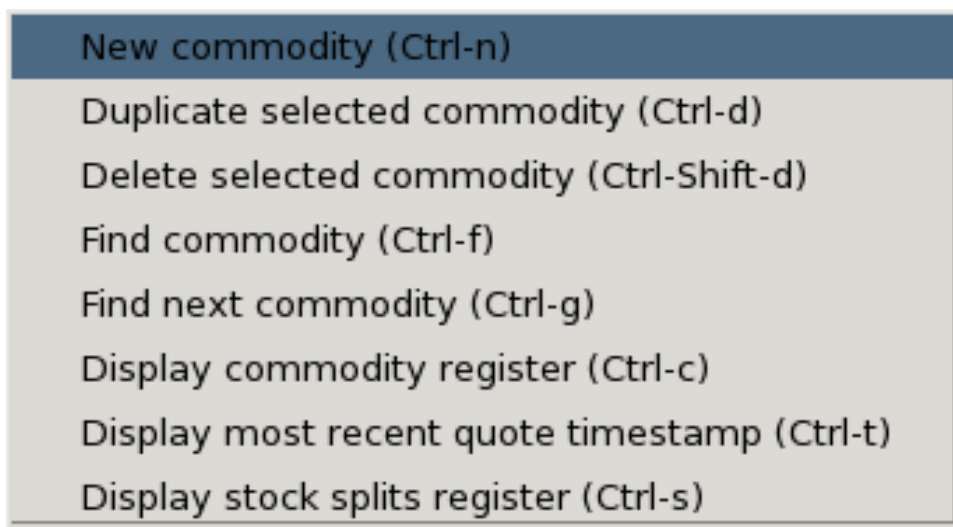
Figure 5.11: Commodities Register Menu

Find Commodity (Ctrl-f) This will cause a dialog to appear (see Figure 5.12), allowing you to search for a commodity. This is useful if you have defined a lot of commodities. The search is done forward, beginning with the commodity at the top of the commodities window. There is a pulldown in the dialog that allows you to specify the column in the commodities window that you wish to search. The "Find expression" is a so-called "Regular expression". For a brief description of the regular expression pattern-matching language, see Section A.

Find Next Commodity (Ctrl-g) To use this command, you need to have done a previous Find Commodity. Find Next Commodity will re-apply the regular expression to the column specified in that Find, starting with the next commodity (moving down in the commodities window).

Display Commodity Register (Ctrl-c) To use this command, you must have a commodity selected in the commodities window. It will display the commodity register for that commodity; see Figure 5.13. As you can see, the commodity register simply provides the pricing history for the selected commodity.

Display most recent quote timestamp (Ctrl-t) This command displays the timestamp (date and time) of the most recently-obtained quote across all commodities. Usually, you will obtain quotes in bulk, online, with a script (`quotes.py`) provided with Newcash, or by having that script run as part of the Newcash report generating process, rather than entering quotes manually. If your usual practice is to obtain quotes online, this command
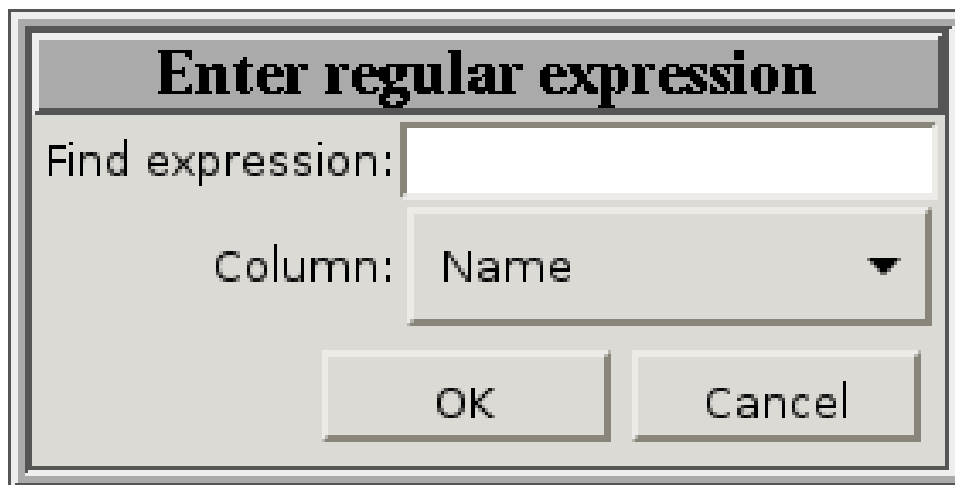
Figure 5.12: Commodities Find Dialog

will tell you when you last did this. Otherwise, it simply gives you the timestamp of the last manually entered quote.

Display Stock Splits Register (Ctrl-s) To use this command, you must have a commodity selected in the commodities window. It will display the stock splits register for that commodity; see Figure 5.15. As you can see, the stock splits register provides the stock split history for the selected commodity.

5.5.4 Commodity Register

Commodity Register Fields

Timestamp The date and time of the quote. This field is editable. All the date-editing operations that are available in Account Registers are also available here. See Section 5.5.1.

Price

Commodity Register Operations

New quote (Ctrl-n) Insert a new blank quote having the current date and time as the time-stamp. You can then edit the Price field to manually insert a quote.

Delete selected quote (Ctrl-Shift-d) Select the quote to be deleted before running this command.

Apple	
Time-stamp	Price
2016-06-07 14:45:33	99.4699
2016-05-12 23:31:23	90.3400
2016-05-03 12:18:56	94.5299
2016-04-22 08:03:17	105.9700
2016-04-14 14:07:52	112.0000
2016-03-01 18:11:08	100.5300
2016-01-22 17:38:47	101.4200
2016-01-07 23:47:59	96.4500
2015-12-21 23:42:28	107.3300
2015-11-22 21:55:45	119.3000
2015-10-31 15:32:14	119.5000

Figure 5.13: Commodity Register

New quote (ctrl-n)
Delete selected quote (ctrl-shift-d)
Display calendar for selected transaction (Ctrl-a)

Figure 5.14: Commodity Register Menu

Figure 5.15: Stock Splits Register

Note that there is no "Display calendar ..." operation for this register. I've omitted it because I have never found a need for it. In the vast majority of cases, I get quotes using the -q option of the Report Generator, which obtains the quotes online and, of course, they are time-stamped. On the rare occasions when I enter a quote manually, the default time-stamp (the date-time of entry of the quote) is exactly what I want. If in the future a need arises for this, I will add it. But for now, I prefer not to litter the code with a solution in search of a problem.

5.5.5 Stock Splits Register

Stock Splits Register Fields

Split Date The date of the stock split. This field is editable. All the date-editing operations that are available in Account Registers are also available here. See Section 5.5.1.

Split Factor The factor (or multiple) by which the stock split. If we call the factor 'f', the new shares outstanding 'n' and the old shares outstanding 'o', then $f = n/o$

Total Split Factor The product of the factor of the row in question and all the rows above it.

Stock Splits Register Operations

New stock split (Ctrl-n) Insert a new blank stock split having the current data. You can then edit the Factor field to complete the entry.
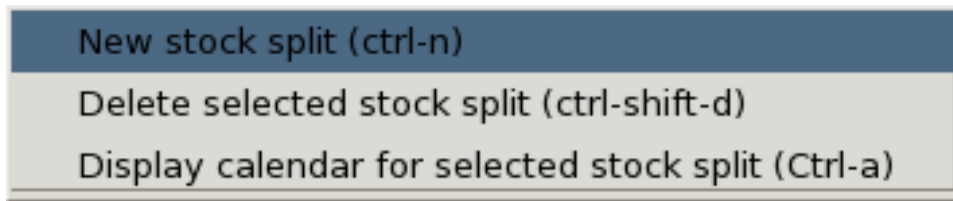
Figure 5.16: Stock Splits Register Menu

Delete selected stock split (Ctrl-Shift-d) Select the stock split to be deleted before running this command.

Display calendar for selected stock split (Ctrl-a) displays a calendar, allowing you to alter the date of the selected stock split.

5.6   Tools/Utilities

My work on the software described in this document was done with the "Unix philosophy" in mind: programs should do one thing and do them well, together with a means for composing their effect. Accordingly, I have built a suite, a collection, of programs, each of them performing a specific function, with the Newcash database serving as the means composition. I think this is preferable to a single monolithic application that tries to do everything and more (the "Bloatware philosophy", invented by Microsoft). An advantage of this approach is that each component of the suite can be written in the programming language that best suits the task at hand[8].

A few things I would like to note before discussing the utility programs:

Naming convention When I speak of "Newcash" (or the actual file-name of the Unix executable, `newcash`), I refer to the main program, to which this document has been addressed thus far. Each utility described in this section has its own name. When I have the need to refer to the whole set of programs, I will use the term "Newcash Suite".

Newcash operations In the following discussions, I will refer to various Newcash operations (adding a new account, transaction, or split, pasting a split account, etc.) without explicitly showing you how to perform them. All Newcash operations are documented in Sections 5.4.1, 5.5.1, 5.5.2, 5.5.3 and 5.5.4. I have also provided tutorials/examples on their use in Section 5.7.

Formatting commands in this document There are a number of examples of things you must type to a shell, or enter in a file. Sometimes these

---

[8]Monolithic systems can also be written in a variety of languages, but doing so is more difficult than in the case of a suite of separate components.

examples are long enough that, without some adjustment, they would exceed the standard width of this text, as determined by the system I used to produce this document, Latex. I could have used smaller fonts in these cases, but they would have had to be impossibly small, affecting readability. Instead, I chose to use a standard technique used in Unix that allows you to write commands on multiple lines and have them interpreted as if they were one: where I need to break a line, I introduce a back-slash (\) before the new-line character. You should interpret the back-slash/new-line two-character sequence as the Unix shell does, treating it as if it were a single space character. The point is to make the command more readable without changing its meaning.

Tcl Some of the utilities are written in Tcl[9]. Because Tcl is an "interpreted" language, the utilities written in it depend on the presence of certain packages in your environment[10]:

- You must insure that the Tcl package is installed on your system.

- The Newcash utilities need to access your Newcash Sqlite database. The Tcl Sqlite library must therefore be installed and generally it is not installed with the basic Tcl package. The name of this package varies by system. Here are a few examples:

  Debian-Ubuntu-Mint: libsqlite-tcl
  OpenSuse-OpenBSD: sqlite3-tcl
  FreeBSD-DragonFly: tcl-sqlite3

- In order to obtain quotes on your security positions, you will need Tcl's HTTP support, which is usually supplied by your system's `tcllib` package.

Another issue: when you run the utilities written in Tcl, because they are interpreted programs, the proper interpreter, called `tclsh` must be available in one of the directories in your `PATH` variable, so that it can be found. Some systems, such as Debian Linux, install the Tcl interpreter as `/usr/bin/tclsh`. Because Linux stores executables for programs that are part of the distribution (e.g., `cp`, `ls`) in the same directory (`/usr/bin`) as the executables supplied by packages, this directory is certain to be in your PATH and you will have no trouble running the utilities.

Other systems, such as FreeBSD, install the Tcl interpreter as

`/usr/local/bin/tclsh8.6`

This presents two problems:

---

[9]Tcl (Tool Command Language), a scripting language, was originally developed by Prof. John Ousterhout and his students at the University of California, Berkeley.

[10]Programs written in compiled languages can be delivered in self-contained executable files, using a technique called "static linking".

1. The filename of the interpreter is not `tclsh`, which is what the utilities require, and

2. The executable is in a directory that may not be in your PATH.

You can solve both of these problems by creating a `bin` directory in your home directory (if you don't already have one) and creating a symbolic link in that directory to the Tcl interpreter executable file, naming the link `tclsh`. You can accomplish this using the commands

```
mkdir ~/bin # Needed only if it doesn't exist
cd ~/bin
ln -s /usr/local/bin/tclsh8.6 tclsh
```

You must then add your bin directory to your PATH variable. That environment variable is usually set in your `~/.profile` or `~/.bash_profile`. Edit the correct file to add your `bin` directory to the front of the colon-separated directory list to which the PATH variable is set. You can, of course, always check the current setting of PATH with the command `echo $PATH`.

5.6.1   Scheduling Recurring Transactions

Recurring transactions are transactions that need to be entered periodically and are substantially the same each time. The Newcash Suite includes two utility programs

`newcashScheduledTransaction`

and

`newcashScheduledLoanPayment`

that, together with `cron`, a tool supplied by the Unix/Linux environment, will enter these transactions automatically on a schedule you specify. Newcash supports two categories of recurring transactions:

- Transactions in which the split amounts remain constant with each recurrence.

- Loan payments of the kind that are annuities from the point of view of the lender (e.g., home mortgages, automobile loans) and therefore have a constant total payment amount, but the amounts flowing to principal and interest vary with each payment.

You make use of this capability by describing, to one of the afore-mentioned utilities, an existing "template" transaction that you want replicated periodically. You then arrange for cron to run the utility on a schedule that insures that the replications occur with the correct frequency.

Scheduling Recurring Non-Loan Transactions

To set up the scheduled replication of non-loan transactions, you will arrange for cron to run the Newcash application `newcashScheduledTransaction` periodically. `newcashScheduledTransaction` requires five command-line arguments, as you can see from its "usage" message, which you get if you simply invoke the program from a shell without any arguments:

```
Usage: newcashScheduledTransaction date num description\
    minimum-period path-to-database
```

The values of the first three arguments are used to identify an existing transaction, the so-called "template" transaction. The template transaction described by those arguments must exist and be unique.

If these criteria are satisfied, a new copy of the template will be inserted into the database. The copy is not exact. The transaction date will be the date on which `newcashScheduledTransaction` is run. The template's splits' reconciled and transfer flags are not copied. The new transaction is otherwise the same as the template.

If the first three command-line arguments do not describe an existing transaction, `newcashScheduledTransaction` will issue the error message "Specified template transaction not found.". If the three arguments identify more than one transaction, `newcashScheduledTransaction` will issue the error message "Template specification matches more than one transaction." . In either case, the program will then quit, having done nothing.

The minimum-period argument defines the minimum time that must elapse, in days, before `newcashScheduledTransaction` will again copy a template that has already been copied. So if you run `newcashScheduledTransaction` on a certain template transaction on 2014-06-01 and it succeeded, and then ran it on the same template every day subsequently specifying a minimum-period of 28, it would not have replicated the template again until 2014-06-29.

Let's look at an example, your paycheck. Assume that the date of the manually-entered paycheck transaction that you will use as a template is "2013-05-15", the num field is "PAYCHK", and the description field is "Paycheck". This transaction can be seen in the Checking Account register, as shown in Figure 5.17. This transaction's splits can be seen in Figure 5.18.

If your Newcash database file is in `~/Finances/Finances.newcash` and you arrange for `cron` to run the command

```
newcashScheduledTransaction 2013-05-15 PAYCHK\
    'Paycheck' 28 ~/Finances/Finances.newcash
```

on 2013-06-15, you will get an almost exact replica of the template transaction. The result can be seen in Figures 5.19 and 5.20.

Now let's look at how to get the above command run periodically by `cron`. If you wish to run `newcashScheduledTransaction` on the fifteenth day of every month, you might make an entry in your crontab file something like this:

Figure 5.17: Paycheck Template Transaction

Figure 5.18: Paycheck Template Splits

Figure 5.19: Second Paycheck Transaction

Figure 5.20: Second Paycheck Splits

```
PATH=/home/<your login name>/bin:/usr/local/from_source/bin:/usr/local/bin
#
#minute hour mday month wday command
00      8-22    15       *        *        newcashScheduledTransaction \
                                           2013-05-15 PAYCHK \
                                           'Paycheck' 28 \
                                           $HOME/Finances/Finances.newcash
```

Note the setting of the PATH environment variable. The above suggestion will work on either Linux or BSD systems, assuming that on a BSD system, you have provided a symbolic link to `tclsh` in your `bin` direcotory, as discussed earlier.

Next, you need to register your crontab file with cron. Let's assume the above cron directive is in the file `~/cron/my_crontab`. The `crontab` command will make `cron` aware of your file:

```
crontab ~/cron/my_crontab
```

You need to be sure you have permission to use cron. It is likely that the information you need can be obtained with the command

```
man crontab
```

If your system provides the widely-used Vixie cron, you will need to add your user-name to a file with a name similar to `cron.allow` (the crontab man page will tell you what the exact file-name is and where it is located). You will need super-user privileges to do this, since it involves writing a file in a system directory that is not writeable for ordinary users.

See `man 5 crontab` for a fuller explanation of the format of the crontab file than I have provided here.

66

Please note that the above example crontab entry specifies that the utility program is to be run every hour on the hour from 0800 to 2200 on the fifteenth day of every month. You are probably asking "why run the program multiple times to make one entry?". The reason is that you may be running Newcash at the same time the cron job runs. Both Newcash and the utility program have write access to your Newcash database and Sqlite is very conservative about locking the database to avoid problems caused by simultaneous writers. So it is possible that an attempt to run `newcashScheduledTransaction` will fail due to a lock collision. Running it multiple times helps to insure that it succeeds. But if it succeeds, won't subsequent runnings cause duplicate transactions to be entered? No, because of the minimum-period logic described earlier.

Also note that if you have multiple cron jobs that fire on the same day and hour, all of them using your Newcash database, it's a good idea not to schedule them all to run at the same minute (and thus at exactly the same time). If you do, they will compete with each other for the database lock and some will fail due to the database being locked. Offsetting them by a minute or two gives each a chance to complete before the next one starts, maximizing the chance of getting all your jobs run early in the schedule you have specified.

Scheduling Loan Payment Transactions

When you borrow money to buy a home or a car (or lease a car, which is a form of loan with a non-zero final value), which are assets, money flows from you and your lender to purchase that asset. You need to set up a Liability account for the loan and an Asset account for what you bought.

As with non-loan transactions, automatically entering loan transactions requires the use of a "template" transaction. In this case, the template transaction should be the first payment you make on your loan. That transaction must have (at least[11]) the following splits:

- A split for the money flow from the account used to make the loan payment. The account is typically your checking account. The amount is the negative value of the total payment (negative because the money is flowing away from the account).

- A split for the principal payment. The account will be the Liability account you set up to represent the loan.

- A split for the interest payment. You will need to set up an Expense account for the loan's interest expense.

But in order to properly enter your first loan payment transaction, you need to know what the principal and interest amounts are. I have provided

---

[11]If the loan is a home mortgage, your lender might require that you make real-estate tax payments to them, which they hold in "escrow" until the payment is due. If so, you will need an additional split to account for that money flow. Either way, you will need an Expense account for your real-estate taxes.

a Libreoffice spreadsheet[12] that will allow you to enter the terms of your loan (principal borrowed, annual interest rate – don't forget the trailing "%" sign! – the term, number of payments/year, the future value – $0 in the case of an ordinary mortgage or automobile loan, but the residual value if you are setting up an automobile lease). You will find that spreadsheet in

```
newcash/utilities/newcashLoanPayment.ods
```

in the `newcash` directory that contains your local copy of the Newcash repository that you cloned from GitHub. The spreadsheet will calculate the correct initial interest and principal payments for you, for use when you enter the transaction for that payment.

When you create the transaction for your first loan payment, you will, as described above, enter splits for the total payment, principal and interest money flows. In the memo field of each of those splits, you will need to enter something that is descriptive of what the split is for, for use by the `newcashScheduledLoanPayment` utility, as explained below. Obvious candidates are "Payment", "Principal" and "Interest" in the respective splits. If you have multiple loans, you don't need to worry about making these memo fields unique[13] across all of them. You can use the same suggested names in the memo fields of the splits for the template transactions for each your loans. The memos simply need to uniquely identify what each split is for within each template transaction.

To set up the scheduled replication of loan transactions, you will arrange for cron to run the Newcash application `newcashScheduledLoanPayment` periodically. `newcashScheduledLoanPayment`gc requires ten command-line arguments, as you can see from its "usage" message, which you get if you simply invoke the program from a shell without any arguments:

```
Usage: newcashScheduledLoanPayment date num description \
principal-memo interest-memo payment-memo \
annual-interest-rate payments-per-year \
minimum-period path-to-database
```

The values of the first three arguments are used to identify an existing transaction, the so-called "template" transaction. The template transaction described by those arguments must exist and be unique. The next three arguments are the contents of the split memo fields that identify the purpose of each split, as described above. The next argument, the seventh argument, is the annual interest rate expressed as a percentage, but *without* a trailing % sign. The eight argument is the number of payments per year,

The minimum-period argument defines the minimum time that must elapse, in days, before `newcashScheduledLoanPayment` will again copy a template that

---

[12]Libreoffice is an open-source Office-like system that you can install with the package menager on whatever Linux or Unix system you are using.

[13]By "unique" I mean that there must not be more than one transaction in your Newcash database having the same post-date, num and description fields. The utility will fail if your specified template is not unique.

Figure 5.21: Home purchase transaction

has already been copied. So if you run `newcashScheduledLoanPayment` on a certain template transaction on 2014-06-01 and it succeeded, and then ran it on the same template every day subsequently specifying a minimum-period of 28, it would not have replicated the template again until 2014-06-29.

If the template transaction is found, based on the first three arguments, and the split memos match properly, a new copy of the template will be inserted into the database. The copy is not exact, of course. The transaction date will be the date that `newcashScheduledLoanPayment` is run. The template's splits' reconciled and transfer flags are not copied, nor are the memo fields, and the principal and interest values are calculated based on the current principal balance. The new transaction is otherwise the same as the template.

If the first three command-line arguments do not describe an existing transaction, `newcashScheduledLoanPayment` will issue the error message "Specified template transaction not found.". If the three arguments identify more than one transaction, `newcashScheduledLoanPayment` will issue the error message "Template specification matches more than one transaction.". If the split memos do not match the three memo arguments given to the command, an error message will be issued, complaining that a query failed to return a row. In all these cases, the program will then quit, having done nothing.

Let's look at an example. Suppose you bought a home for $220,000, with the closing on 5/1/2013. Further assume that you provided $20,000 toward the purchase and financed the rest with a 30-year, $200,000 mortgage at an annual interest rate of 4%. To record the transaction with Newcash, you will need a new Asset account for the home and a new Liability account for the mortgage loan. You then enter a transaction for the purchase with three splits, as shown in Figure 5.21. The transaction has splits as shown in Figure 5.22. As you can see from the splits, I am assuming that the $20,000 you supplied came from your savings account.

One month later, you have to make your first mortgage payment. The lender will send you a bill for the amount of the payment, but, in my experience, they

2013-05-01 Purchase home				
Account	Memo	R	Value	Balance
:Liabilities:Home mortgage		☐	-200000.00	-200000.00
:Assets:Home		☐	220000.00	20000.00
:Assets:Bank Accounts:Savings		☐	-20000.00	0.00

Figure 5.22: Home purchase splits

are unlikely to tell you how much of that payment goes to interest and to principal. If my experience proves correct, you will need to use the mortgage calculation spreadsheet I have provided to obtain the principal and interest payment amounts for the first payment. You will find the spreadsheet in the file

`newcash/utilities/newcashLoanPayment.ods`

To use the spreadsheet, you will need the Libreoffice package installed on your system. Figure 5.23 shows the spreadsheet with the terms of our example entered. With this information, we are ready to enter the transaction for the first mortgage payment. I assume you have created an

`Expenses:Home:Mortage interest`

account. To create the transaction, you need to open a register associated with any of the accounts involved in this transaction. Typically, you will pay your mortgage with your checking account, so entering the transaction via the checking account register makes sense. See Figure 5.24 to see what the Checking Account register looks like prior to entering the first mortgage payment. Next we enter the mortgage payment transaction. But we haven't yet worked on its splits, so the splits will be those provided by Newcash by default for a new transaction. See Figures 5.25 and 5.26. Now we edit the splits by adding a third, correcting the accounts for the principal and interest money flows, setting their memo fields for the `newcashScheduledLoanPayment` utility, and setting the amounts of each split's money flow (once two of the three amounts are entered, using the Transaction Register "Balance transaction" command with the third selected will set its amount correctly). See Figure 5.27 to view the splits after editing.

If your Newcash database file is in `~/Finances/Finances.newcash` and you run

```
newcashScheduledLoanPayment 2013-06-01 MortPmt\
   'Mortgage payment' 28 ~/Finances/Finances.newcash
```

70

Figure 5.23: Mortage Example Spreadsheet

Figure 5.24: Checking account before entering first mortgage payment

Figure 5.25: First mortgage payment transaction with default splits

Figure 5.26: First mortgage payment, initial default splits

Figure 5.27: First mortgage payment, edited splits

:Assets:Bank Accounts:Checking					
Date	Num	Description	R	Value	Balance
2013-05-15	Paychk	Paycheck	☐	3420.00	3420.00
2013-06-01	MortPmt	Mortgage payment	☐	-954.83	2465.17
2013-07-01		Mortgage payment	☐	-954.83	1510.34

Figure 5.28: Second mortgage payment transaction

2013-07-01 Mortgage payment				
Account	Memo	R	Value	Balance
:Liabilities:Home mortgage		☐	289.12	289.12
:Expenses:Home:Mortgage interest		☐	665.71	954.83
:Assets:Bank Accounts:Checking		☐	-954.83	-0.00

Figure 5.29: Second mortgage payment splits

on 2014-07-01, you will get an almost exact replica of the template transaction, as shown in Figures 5.28 and 5.29.

If you wish to automatically run `newcashScheduledLoanPayment` on the first day of every month to enter your mortgage payment, you would make an entry in your crontab file something like this:

```
PATH=/usr/local/from_source/bin:/usr/local/bin:$PATH
#
#minute hour mday month wday command
00      8-22  01    *     *    newcashScheduledLoanPayment \
                                  2013-06-01 MortPmt \
                                  'Mortgage payment' 28 \
                                  $HOME/Finances/Finances.newcash
```

See the discussion in at the end of the previous sub-section ("Scheduling Recurring Non-Loan Transactions") regarding crontab file entries.

5.6.2   Report Generator

The Newcash Report Generator is a substantial application in its own right, called `newcashReportGenerator`. It reads your Newcash database to produce a Balance Sheet report, an Income and Expense Statement, a statement of your Net Worth, and then several reports about your open investment positions. The reports are written by newcashReportGenerator to a file of your choosing in LATEXformat[14]. They must then be processed by LATEXto produce the final product, usually a .pdf file.

Because the newcashReportGenerator takes quite a few command-line arguments and options and then its LATEXoutput must be post-processed to produce the actual reports in .pdf format, I have provided a utility, called newcash-GenerateReports, to make report generation easy. This script provides sensible defaults for all the arguments required by the report generator. Optionally, before running the report generator, it will run a Newcash utility to obtain quotes on all your open positions, so the asset values in the Balance Sheet and the unrealized capital gains in the Investments reports are up-to-date. After running the report generator, the script runs pdflatex to generate the final product, a .pdf file. If you run the command

`newcashGenerateReports -h`

from a shell, you will see the following options described:

-f is the path to the Newcash database from which you wish to generate the reports. Default is `~/Finances/Finances.newcash`.

---

[14]LATEXis a document preparation system originally developed in 1984 by Leslie Lamport. It is a layer on top of TEX, the typesetting system developed by (now-retired) Stanford Professor Donald Knuth, a greatly respected figure in Computer Science and a Turing Award winner. Knuth wrote a now-classic work called "The Art of Computer Programming". He developed TEXbecause there was no typesetting system available at the time that he considered satisfactory. The document you are reading was produced using LATEX.

Assets					433367
Home				350000	
Investments				82728	
Equities and derivatives			79098		
Vanguard		79098			
Tesla Motors	79098				
Cash and cash equivalents			3630		
Vanguard		3630			
Checking				639	
Liabilities					-279130
Home mortgage				-279038	
Credit Card				-92	

Figure 5.30: Balance Sheet

-w is the directory where you want the report files to be written. Default is `/tmp`.

-n is the name you want the report files to have. Default is newcashReports. The extension will be .pdf.

-b is the begin date for the Income and Expense Statement in YYYY-MM-DD format. Default is the current day less one year plus one day.

-e is the end date for the Income and Expense Statement and the date of the Balance Sheet. Default is the current day.

-d is the maximum depth to which to descend in the account tree when generating the Balance Sheet and Income and Expense Statement. Default is 5.

-q takes no argument. If present, quotes for open positions will be obtained before generating the reports.

-s takes no argument. If present, report generation is skipped.

-h generates this usage message.

Balance Sheet

The Balance Sheet is a snapshot of your assets (what you own) and liabilities (what you owe) at a moment in time, in this case the end date newcashGenerateReports passes to the Report Generator. The Assets are presented in descending order of their value, so your most valuable assets will be toward the top of the report. The Liabilities are sorted by their current balance in

2 Income and Expense Statement (2013-04-30 through 2015-08-03)

Income					-12000
Salary				-12000	
Expenses					2194
House				2096	
Mortgage interest			2096		
Utilities				92	
Electricity			92		
Investments				5	
Commissions			5		

Figure 5.31: Income and Expenses Statement

ascending order, because liability accounts tend to have negative balances, as discussed earlier in this document. For example, if you have a liability account with a balance of -$100000, meaning that you owe that lender $100000, and another liability account with a balance of -$5, the former is obviously the more important liability and you want to see that at the top of the liability portion of the Balance Sheet. The ascending order by balance accomplishes that.

The number of levels displayed, beginning with

`:Root:Assets`

and

`:Root:Liabilities`

will be less than or equal to the "maximum depth" you provide when you run newcashGenerateReports, either by indicating a value with the -d option, or using the default value. I say "less than" because there may not be "maximum depth" levels in the Assets and/or Liabilities sub-trees.

See Figure 5.30 for an example Balance Sheet, generated from the example Newcash database, which you will find in

`newcash/documentation/examples.newcash`

Income and Expense Statement

The Income portion of the Income and Expense Statement shows the total flows in the accounts starting with

`:Root:Income`

and descending a number of levels into the account tree equal to the maximum depth argument you specified to newcashGenerateReports. Reminder: it is perfectly normal in Newcash for Income accounts to have negative balances. Their

reason for being is to provide income and so flows tend to be away from them, usually to Asset accounts, such as a savings account (e.g., the receipient of interest income), a brokerage's cash account (e.g., the recipient of cash dividends), or shares of stock or a mutual fund (e.g., the recipient of reinvested dividends). You will note that because of the normal negative sign of Income accounts, the report is sorted in ascending arithmetic order, so the most negative accounts, the ones that contributed the most income, appear at the top of the report.

The Expenses side of the report is similar, except it shows the accounts sub-tree starting with

`:Root:Expenses`

Expense accounts tend to be positive; money flows to them, usually from asset accounts. When you write a check to your electric company, money flows from your checking account (an Asset account) to, say,

`:Root:Expenses:Utilities:Electricity`

an Expense account.

See Figure 5.31 for an Income and Expense Statement taken from the example Newcash database.

Net Worth Statement

Your Net Worth (or Equity), which equals the total current value of your assets plus your total liabilities, is the true measure of your wealth. The Newcash Report Generator calculates your Net Worth for you.

It is true that you can have very large positive Net Worth and still have trouble paying your bills, even if your expenses aren't out-of-line with your wealth. This will occur if too small a portion of your assets are liquid – cash or cash equivalents. But if this is the case, you can borrow against your non-liquid assets, such as real estate, to improve your cash position. With a large positive Net Worth, you will be able to borrow at reasonable interest rates. Companies do this all the time to smooth out the ebbs and flows of their cash position and this borrowing is not necessarily a sign of mis-management. But if you have trouble paying your bills and have only a slightly positive or even negative Net Worth, you will find yourself paying a high price (interest rate) to borrow money, or even unable to borrow at all, to get you through cash famines.

Investments

This section presents multiple looks at your open investment positions and how they are faring.

Open Positions This section simply lists your open investment (marketable) positions, giving total shares and current market value (as of the last time you obtained quotes), sorted in descending order of value. Understand that this and the other investment reports are per-commodity, not

per-account. By that I mean that if you own 100 shares of IBM in one brokerage account and 150 shares of IBM in another account, this report will display an aggregate 250-share position in IBM.

Capital Gains As the name implies, this section displays the capital gains, in currency terms of all your open positions, in descending order of the gain. The basis for the gain is calculated, in simple cases, by finding the most recent point at which you held zero shares of a marketable commodity and using the cost of the next opening transaction as the basis. There are more complicated cases that are described in a big comment in the code. If you really need to understand how this works in detail, see `newcash/reportGenerator/newcashReportGenerator.hs`.

Annualized Return on Capital Gains This section is related to the previous one, but displays the gain as an annualized return, in descending order of that return, rather than simply the gain in the currency you use. This is a better measure of the performance of your investments, because it takes both time and how much is invested into account. Obviously, a return of $5000 on a position that originally cost $100000 and that has been held for 10 years is not as good as a $5000 return on a position that originally cost $20000 and has been held for six months.

Capital Gains + Dividends/Interest The Newcash script, newcashLinkIncomeToCommodities, as the name implies, associated income accounts with the commodities that generate the income. This allows the generation of this report, which adds your dividends and interest to the capital gains of your open positions, to give a more accurate view of your positions if you hold income-generating securities.

Annualized Return on Capital Gains + Dividends/Interest This is similar to the previously described Annualized Returns report, but takes income as well as capital gains into account in calculating the return.

Tax Report

Despite the expectation raised by the title of this section, the Newcash Report Generator does not produce a separate tax report. Instead, tax reporting is integrated with both the Balance Sheet and Income and Expense reports, though the latter is likely to be of much greater use to you. Any accounts that inherit the "descendents are tax-related" are property are displayed in italics in these reports. Typically, those will be Income or Expense accounts, which is why I just suggested that the Income and Expense report is likely to be more useful than the Balance Sheet at tax time. To take advantage of this, simply run the Report Generator with a start-date of the beginning of the tax year (e.g., 2014-01-01) and an end-date at the end of the tax year (e.g., 2014-12-31). The italicized portions of the resulting Income and Expense report will provide you with the information you need for filing your income taxes. Obviously, it is your

responsibility to organize your accounts in the Newcash database in sensible fashion and correctly designate those portions of the account tree as tax-related by adding the "descendents are tax-related" property to accounts that are close ancestors of the tax-related accounts.

For example, you might have the accounts

```
Income:Salary
Income:Salary:John
Income:Salary:Mary
```

where both John and Mary's salaries are taxable and therefore want to be designated as such. To accomplish that, edit `Income:Salary` and check the "Descendents are tax-related?" check-box.

5.6.3   Composite Register

Newcash is designed to provide separate registers for transactions (account registers) and their splits (transaction registers). When viewing an account register, you can inspect the splits of individual transactions by selecting the transaction of interest and invoking the "Display transaction register" command either via menu or keyboard, as discussed in Section 5.5.1. But sometimes it is useful to see a group of transactions displayed, together with their splits. An example is when you use the Statement Importer (see Section 5.6.4 to import a credit card statement. The Statement Importer uses "fuzzy-matching" techniques to try to assign expense accounts correctly, based on inspection of prior transactions, while trying to ignore small differences in the transaction descriptions. This works well, but not perfectly. Also, there are times when the expense account of a new credit card transaction needs to be different than that of a matching earlier transaction, because circumstances have changed.. It is therefore a good idea to inspect the Statement Importer's expense account assignments, which means looking at the splits of a possibly large group of transactions. This is cumbersome to do with Newcash itself. Hence the existence of `newcashCompositeRegister`.

You invoke this utility at the command line as the usage message indicates:

```
Usage: newcashCompositeRegister
-b is the begin date for transactions to be included
       in the composite register. Defaults to first day
       of last month.
-e is the end date for transactions to be included
       in the composite register. Defaults to last day
       of last month.
   -d is the description filter. It defaults to %,
       which matches any description.
   -c is the concise option. When specified, the splits
       that refer to the account selected by the required
       argument below are not included in the output.
```

```
-h generates this usage message.
    The above options must be followed by two required arguments:
        the path to the account for which you want the
        composite register, and the path to the Newcash database.
```

The two data arguments must be specified in the form `YYYY-MM-DD` (IS0-8601 format). The "description" argument can further filter the transactions you wish to see. The "

The output of the program is to its standard output, so if you want the output in a file, use the usual shell redirection notation.

The format of the output is each transaction found, followed on subsequent lines by its splits. The fields of the transactions and the splits are separated by "—" characters, with the splits indented. The intention is that this output will be written to a file that is then loaded into a spreadsheet program such as Libreoffice, for display.

For example, assume we have copied the example Newcash database to `/tmp` and we run the commands

```
cd /tmp
newcashCompositeRegister -b 2000-01-01 -e 2016-12-31 \
':Assets:Bank Accounts:Checking' examples.newcash > checking.bsv
```

The output file, `checking.bsv` is a vertical-bar-separated text file that can be loaded into Libreoffice. To do so, run Libreoffce and create a new spreadsheet. Then either use the menu to run $File \rightarrow Open$ or use the keyboard accelerator `ctrl-o` to open the `checking.bsv` file you just wrote. Libreoffice will ask you to specify the separator options. Uncheck all except "Other", and enter a vertical bar ("—") in the entry box. Hit ↩ (the Enter key) and the file will be loaded. It will look better if you select the "Value" column and, from the menu-bar, format the column with $Format \rightarrow Number format \rightarrow Currency$ or use the keyboard accelerator `ctrl-shift-4` to accomplish the same thing. See Figure 5.32 to see the appearance of this spreadsheet.

When using this utility to check the work of the Statement Importer when importing a credit card statement, all of the transactions in the composite register will have two splits. One of the splits will refer to your credit card account, the other to an Expense account. The credit card account splits are unnecessary clutter, because every transaction will have one and the value will be the negative of the Expense account split. You can easily filter out those splits before importing the output file into Libreoffice by piping the output of `newcashCompositeRegister` into `sed`, an editor useful in scripts. If your credit card account is named `:Liabilities:American Express`, then a command such as

```
cd /tmp
newcashCompositeRegister -c -b 2013-01-01 -e 2013-01-31 \
':Liabilities:American_Express' Finances.newcash > amex.bsv
```

will produce a bar-separated file with the repetive splits referring the American Express account removed. See Figure 5.33

Figure 5.32: Composite Register – Checking Account

Figure 5.33: Concise Composite Register – American Express

5.6.4 Statement Importer

The Newcash Statement Importer will load transactions provided in OFX format[15]. It is most useful for loading credit card transactions and attempting to properly assign the correct Expense account to each transaction. In my experience, it works very well with transactions downloaded from American Express and from Chase.

The primary problem solved by the Importer is the assignment of the correct Expense accounts to the transactions. The method used is to compare each of the incoming transactions with all of the past transactions involving the Liability account for the credit card in question, and using the Expense account of the best match it can find. The matching is done on the transaction's Description field. The difficulty is that the credit card companies are not particularly careful to insure that repeated transactions involving the same vendor will be described identically, so a simple string comparison will not work. Instead, I used a "fuzzy" string matching technique, computing what is known as the Levenshtein Distance, to compare incoming transaction descriptions with the descriptions of past transactions. This works well, but it is not fool-proof and you will need to inspect the assigned Expense accounts after running the Importer, manually correcting those that need attention. I have found the `newcashCompositeRegister` tool invaluable for doing this checking. See Section 5.6.3.

To run the Importer, issue the command

```
newcashStatementImporter <path-to-statement> <path-to-Newcash-database>
```

For example, if the downloaded statement resides in `/tmp/amex.ofx` and your Newcash database is in `~/Finances/Finances.newcash`, then the command

```
newcashStatementImporter /tmp/amex.ofx ~/Finances/Finances.newcash
```

will load the downloaded transactions.

5.6.5 Verifier

As mentioned earlier, your Newcash data is stored in a Sqlite database. The database has a very specific form, with prescribed inter-relationships among data items. The Newcash application itself does incremental checking of the user's actions to try to prevent the database from becoming mal-formed or, at the least, wasting space. But it does not ever take a step back and periodically inspect the database as a whole, a potentially time-consuming task depending on the size of your database, and catching certain types of errors requires this broader look at the database. I've created a verification utility, `newcashVerifier`, that performs this task. You can run the verifier any time you choose using the command

---

[15]From Wikipedia: "Open Financial Exchange (OFX) is a data-stream format for exchanging financial information that evolved from Microsoft's Open Financial Connectivity (OFC) and Intuit's Open Exchange file formats."

```
newcashVerifier <path-to-your-database>
```

The messages it issues should be self-explanatory. After finishing, it will run the Sqlite `vacuum` command, which, as its name implies, reclaims unused space in the database and does other cleanup tasks.

I would suggest backing up your Newcash database before running the verifier. I do not say that based on any negative personal experiences with it, but since `vacuum` rebuilds the whole database and backing up before using the verifier is easy to do (just make a copy your Newcash database file), I think this is a sensible, low-cost precaution.

5.7   How To

To be written.

Appendix A

Regular Expressions

Regular expressions are an extremely flexible and powerful pattern-matching language, employed by Newcash in the various Find commands. Newcash uses the GRegex facility provided by the Gnome Project's GLib library. For a detailed description of the regular expression language supported by GRegex, see

`https://developer.gnome.org/glib/stable/glib-regex-syntax.html`

Don't be daunted by the complexity of the language. Most of the time, you will use a very simple subset of it. For example, to search for a commodity whose name contains the string "National", you would simply enter that string as the Find expression. The searches are case-insensitive, so "national" would produce the same result.

Getting a little fancier, wild-card matching is easy to do. First, realize that any single character that doesn't have special meaning (any of the alpha-numeric characters, for example) is defined as a regular expression that matches itself. So "a" and "5" are regular expressions that match "a" and "5", respectively. The regular expression "." matches any character. The regular expression "*" matches zero or more occurrences of the previous regular expression. Therefore ".*" matches any string of characters of any length and so "a.*fgx" will match any substring starting with "a", followed by a string of any length, consisting of any characters, followed by "fgx".

Realize that if you do a Find on a field using the expression "foobar", the first field *containing* the string "foobar" will be found. The field needn't begin with the match expression; it simply needs to contain it. For example, if you are searching the Description field in an account register, the descriptions "abcdfoobarefgh", "foobarabcdefgh", and "abdefghfoobar" would all match the regular expression "foobar". If you wish to find instances that begin with "foobar", then you would need to use the regular expression "^foobar". Similarly, if you want to search for instances of a field ending with "foobar", you would need to use the expression "foobar$".

Appendix B

Report Generator: Computing Capital Gains

Computing the (unrealized) capital gain of an open position is a bit complicated. The capital gain is the value of the current shares, which is the signed quantity remaining * current price, less the average price of the opening transactions times the number of share currently held. Everything after the word "less" is called the "basis" and therein lies the complexity.

There are several cases that need to be handled correctly.

1. A long position established by several buys with no preceding closing transactions.

2. A long position established by several buys with preceding closing transactions.

3. A long position established by buys intermixed with sells, perhaps preceded by transactions that closed previous positions.

4. All of the above, but with short sales.

Because of the Newcash convention for the signs of money flows (flows toward an account are positive, away are negative), quantities (number of shares) must be positive for security purchases, negative for sales. This must be true because, for any security transaction

$$share\_price = total\_cost/number\_of\,shares$$

The *share_price* is always a positive quantity, but *total_cost* is positive for buys, negative for sales, from the perspective of the account that represents the security (*not* the cash account that supplies the money for buys and receives the money for sales). Therefore, in order for *share_price* to always be positive, *number_of_shares* must be positive for purchases, negative for sales.

Our objective is to compute the average cost of establishing the current position, taking out any previous closing transactions. For a commodity with open positions, to compute the cost basis, we need to find the latest date when there were zero shares and work forward from that. Once that is known, then

$$P_{basis} = \Sigma(q_i * p_i)/\Sigma(q_i)$$

will compute the average price/share of the basis, which we call P_{basis}. Let's call the current price P_{now} and the current position size Q_{now}. The unrealized gain, G, is:

$$G = Q_{now} * (P_{now} - P_{basis})$$

Examples   Suppose you bought 100 shares of IBM for \$180.

$$P_{basis} = \$18000/100 = \$180$$

If the price of IBM is now \$200,

$$G = 100 * (200 - 180) = \$2000$$

Suppose I had bought 50 shares of IBM for \$180 and 100 shares for \$175. Then

$$P_{basis} = (50 * 180 + 100 * 175)/(50 + 100) = 176.667$$

If I then sold 75 shares, I now have 75 left. If the current price is \$200, then

$$G = 75 * (200 - 176.667) = 1750$$

Suppose I shorted 100 shares of IBM at \$190.

$$P_{basis} = (-100 * 190)/ - 100 = 190$$

If the current price is \$175, then

$$G = -100 * (175 - 190) = 1500$$

But here's a complication: suppose I bought 100 shares of IBM at 175, then sold 80 at 195, then bought 200 at 185. What is P_{basis}? Computing $p[basis]$ clearly has to take into account the sale of the 80 shares, because only 20 shares are left from the first purchase at 175 to serve as a basis for the current position. I think it's clear that

$$P_{basis} = ((100 - 80) * 175 + 200 * 185)/((100 - 80) + 200) = 184.09$$

The general answer seems to be that each sequence of opening transactions creates a P_{basis} up to the next closing transation. If the close doesn't completely liquidate the position and there are further opening transactions, those create another P_{basis} and the total P_{basis} is the weighted average of the two. And this obviously extends to larger numbers of opening sequences interrupted by closes.