Larry Overkamp

CSC 450

Dr. Lopamudra Roychoudhuri

Project Proposal

Project Background

This project is being developed for a client who has for several years operated in her home a before- and after-school child care service for elementary school students. One of her responsibilities is to prepare year-end statements for customers to use as supporting documentation for the child care expense deduction on their income tax returns, a task that has been time-consuming and difficult because she has had to page through dozens of handwritten pages to calculate the totals. The endeavor is complicated by the fact that her two high school-aged daughters provide most of the after-school care, giving them a part-time income opportunity while relieving my client of that duty after finishing her normal workday. But because the income for all three is recorded separately, the calculation process is that much more complicated.

After preparing this year's totals, the client recounted how exasperating the job was. I suggested that a computerized way of dealing with all that information would probably lessen the burden substantially. She agreed and requested a proposal, which is described herein.

The client's strengths are not within the area of information technology. She can get around the Web comfortably, has a Facebook account, and can produce a basic word processing document if necessary. She has no interest in having a solution adapted from existing productivity software—such as a spreadsheet program—because of the learning curve involved and because the user interface for applications created using that approach is rarely intuitive. Therefore, a specialized program is the ideal solution for her needs.

Deliverables

This project is being approached as a custom solution to a client's specific needs. It is, therefore, not a generic solution that might be applicable to a broad number of child care providers.

The proposed solution will provide the means to:

- Record pertinent information about clients (the parents/guardians of the children being supervised).
- Record pertinent information about caretakers, primarily Tax IDs.
- Record the upcoming schedules of each child, as well as the actual care episodes that occur.

- Record the upcoming schedules of each caretaker, as well as the actual time slots worked.
- Record the rates charged to clients so that amounts due can be automatically calculated.
- Record client payments.
- Calculate the amount due each sitter based on the recorded work schedules.
- Produce year-end statements for customers’ income tax purposes for which totals are automatically computed.
- Record business expenses.

Because the application is intended for a single-user setting, it will be a Microsoft Windows form-based application. A relational database will be used to store the data. Appropriate user documentation will be produced.

Project Plan Overview

Below are the tasks associated with the development of this project.

Task	Completion Date	Description
Learn Key Processes	02/05/2011	This step involves reviewing with the client the content and the methods she uses today to record information and calculate the various figures needed to run the child care service. From there, a list of features needed in the application will be prepared.
Update Knowledge	02/12/2011	Since it has been a while since I have coded a solution in Visual Studio, I will need to familiarize myself with the latest set of Windows form controls and other technology now available.
Design Database	02/20/2011	Based on the process review, the various database tables and their relationships will be designed.
Design User Interface	03/03/2011	The forms and controls needed to support the chosen features will be created.
Create Classes	03/10/2011	The classes for objects used in the application will be coded.
Code Procedures	04/03/2011	The code to support business functions will be coded.
Test	04/11/2011	Functional testing of all features will be performed.
Obtain Approval	04/11/2011	Client approval and feedback will be sought, and the application will be updated per the content of that feedback.
Create Documentation	04/28/2011	The user documentation and materials needed to fulfill Capstone requirements will be produced.

Larry Overkamp

Spring, 2011

CSC450

Projects for Computer Science Majors, IT Majors, or CS/IT Minors

Capstone Project Requirement Specification & Management Document

Chapter 1. General Information about the Project

Section 1 Project Identification

The object of this project is the creation of a computer-based application to support a home-based child care facility that provides before- and after-school care for elementary school children. It will be a custom application, tailored to the needs and business practices of the client who has requested this solution.

Starting Date: January 19, 2011

Ending Date: April 28, 2011

Section 2 Business Problem Description

The client must prepare year-end statements for clients that indicate the amount each client paid for child care during the year. Those amounts are used by some clients on their federal income tax returns. Currently, the creation of those statements is a manual process, requiring the calculation of totals from dozens of pages of hand-written records.

Also recorded manually are the schedules of both the children and the sitters. These are the basis for the calculating of payments from clients, payments to sitters, and ultimately the year-end reports.

Section 3 Project Purpose

The custom application will allow the client to record pertinent information electronically, which will then be the basis of automated reporting for tax purposes. It will also simplify the calculation of payments to the client's assistant caretakers.

Section 4 Project Scope

To support the intended purpose, the application will allow the tracking of the children's care schedules. Client charges, which can vary by client and by time of day, will be recorded. Consideration must be made for a variety of client situations, namely, whether there will be multiple responsible parties for a given child, and how payment is split among those parties.

The application will provide a means of recording client payments. It will also provide the means to specify the caretaker in charge for each session, and the payment amount due to the caretakers. If possible, the application should provide the means to record other business expenses.

The basis of the business logic will be supported by utilities for maintaining the various entities used throughout the application. This set of entities includes the clients, the children, and a variety of default values that will ease data entry.

No other business-related functionality will be supported. Common functions like account receivable/payable, inventory control, and detailed payroll are not included because the business is not complex enough to warrant such inclusion. Integration to other systems like financial software is likewise out of scope.

Other than the ability to record some default system values, no tailoring or customization by the user is provided. As it is a custom application—developed per the client's specific business arrangements and practices—it is not intended to be a general-purpose child care system usable for a variety of caretakers.

Chapter 2. Project Requirements

Section 1 Overall Description

1. Product Features

Master File Data and Setup Value Maintenance
Certain master files must be loaded with data in order to use the application's utilities. These include lists of clients and children. System-wide values are also needed. These include standard hours of operation and base rates charged to the clients.

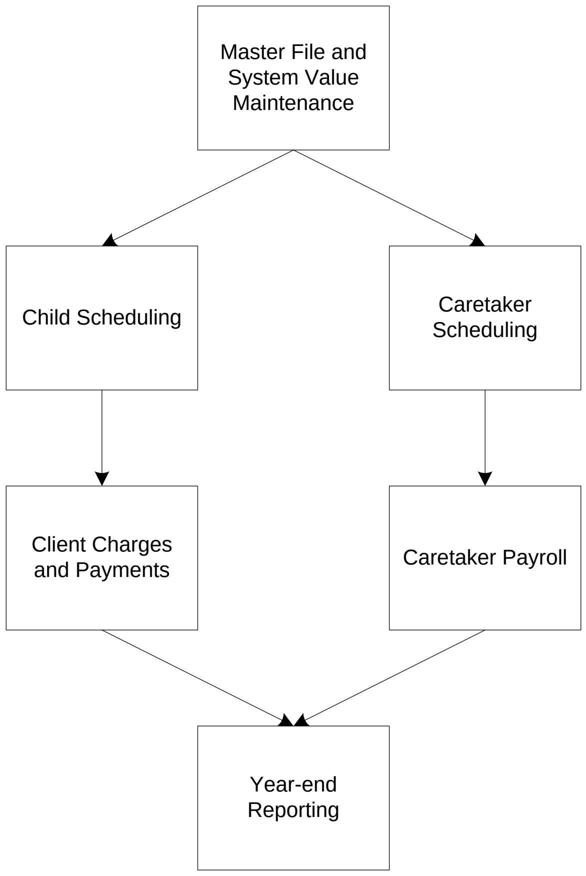
Scheduling
In order to determine the charges for a given client, the application will provide for a means to record each child's day care schedule. Likewise, for compensating the assistant caretakers, a method for recording the caretaker present for each session will be provided.

Invoicing and Payments
Based on actual child attendance, the application will allow the recording of amounts due for each client. Payments for amounts due will also be recordable.

Payroll
Based on caretaker schedules, the ability to calculate each caretaker's pay will be provided.

Reporting
Year-end reporting for income tax purposes—for both clients and caretakers—will be provided.

The diagram below shows the relationships of the application functions described above.



2. Operating Environment

The client for whom this application is being developed uses a Windows XP-based computer, so the application will be developed for that environment. It will be a standalone executable program to be installed on the target computer.

3. Design and Implementation Constraints

Although the client's computer runs Windows XP, it is well-equipped to handle the relatively lightweight requirements of a Windows form application. (The video games that the client's children play on the same computer are of recent vintage, and run without difficulty.) As it is a standalone program run on a single system in the client's home, complex communications and security issues are not a consideration.

The development itself will be with the tools most familiar and readily available to the developer. These include Microsoft Visual Studio 2008, using the C# language, and Microsoft Access for the database.

4. Assumptions and Dependencies

The success of this project depends on a full and accurate description by the client of the business activities. Any omission could jeopardize the project timeline or deliverables.

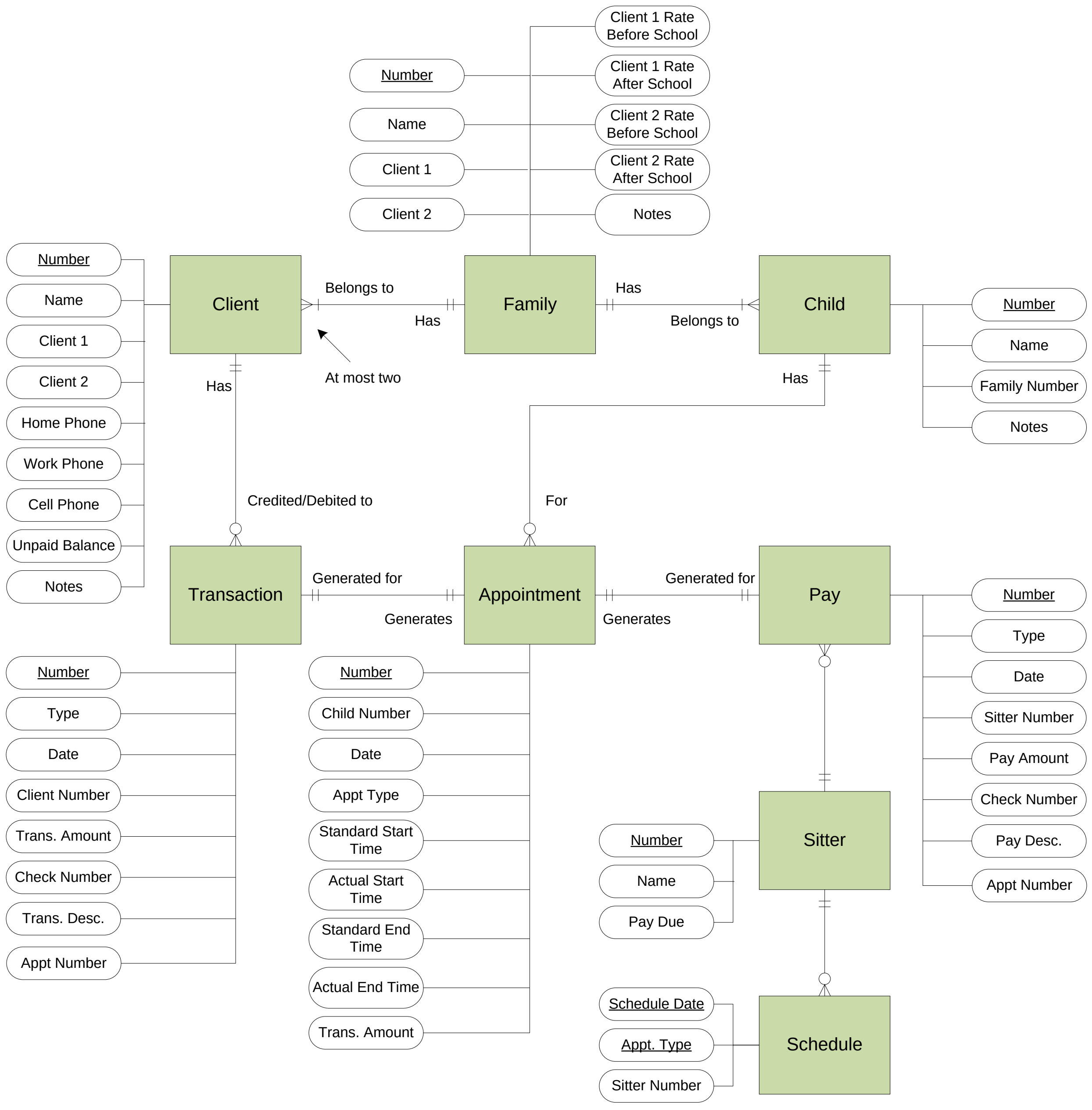
Beyond that, success will rely heavily on the developer's ability to reacquaint himself with the Visual Studio environment—and especially the features and characteristics new to him—and with the C# language. Although these not new to the developer, it has been a few years since he has used them.

Since all components will be created via either Microsoft Visual Studio or Microsoft Access, no dependencies on other software components is expected.

Section 2 System Features

The diagrams that follow are a graphical representation of the features described in this section.

Entity-Relationship Diagram



Entities

Child

This is an individual who will be cared for within the child care setting.

Client

This is an individual who is responsible for paying for a Child’s child care.

Family

This entity associates Child entities and Client entities. This is necessary because multiple Clients may be financially responsible for a given Child.

Appointment

Each episode of care on a given date for a given child is an Appointment. An episode of care can be either before school or after school, so a given child can have at most two Appointments on any given day.

Sitter

An individual responsible for providing care for the Children.

Schedule

A before school time slot or an after school time slot on a given day. There can be at most two Schedule records on any given day.

Transaction

A record of an approved Appointment, which is an Appointment that the user has verified as being completed and for which the appropriate charge amount has been recorded. A given Transaction is for one Client only.

Pay

Also a record of an approved Appointment, but one that identifies the Sitter who was on duty on the date and at the time of the Appointment.

Table Relations

Family/Client/Child

Because more than one individual (client) may be financially responsible for one or more children (as in the case of divorced parents), the Family entity is used to tie Clients and Children together. The Family entity defines the responsibility for each associated Client. There must always be at least one client assigned to a family, and at most two. In a one-client family, there is only one responsible party for that family. This can be the only parent or guardian, or one of two parents or guardians (as in a married couple). For the latter, it is necessary to specify only one parent or guardian. The two-client family takes into account situations such as both parties of a divorce being responsible in part for the family.

A given family can have one or more children.

Appointment/Child

Each Appointment is for one Child only. If more than one Child in a Family are to be present at the same time, they each have their own Appointment record for that time slot. A given Child can have any number of Appointments.

Appointment/Transaction

When an Appointment is approved, a Transaction record is created for each Client who has responsibility for the Appointment's Child.

Appointment/Pay/Sitter

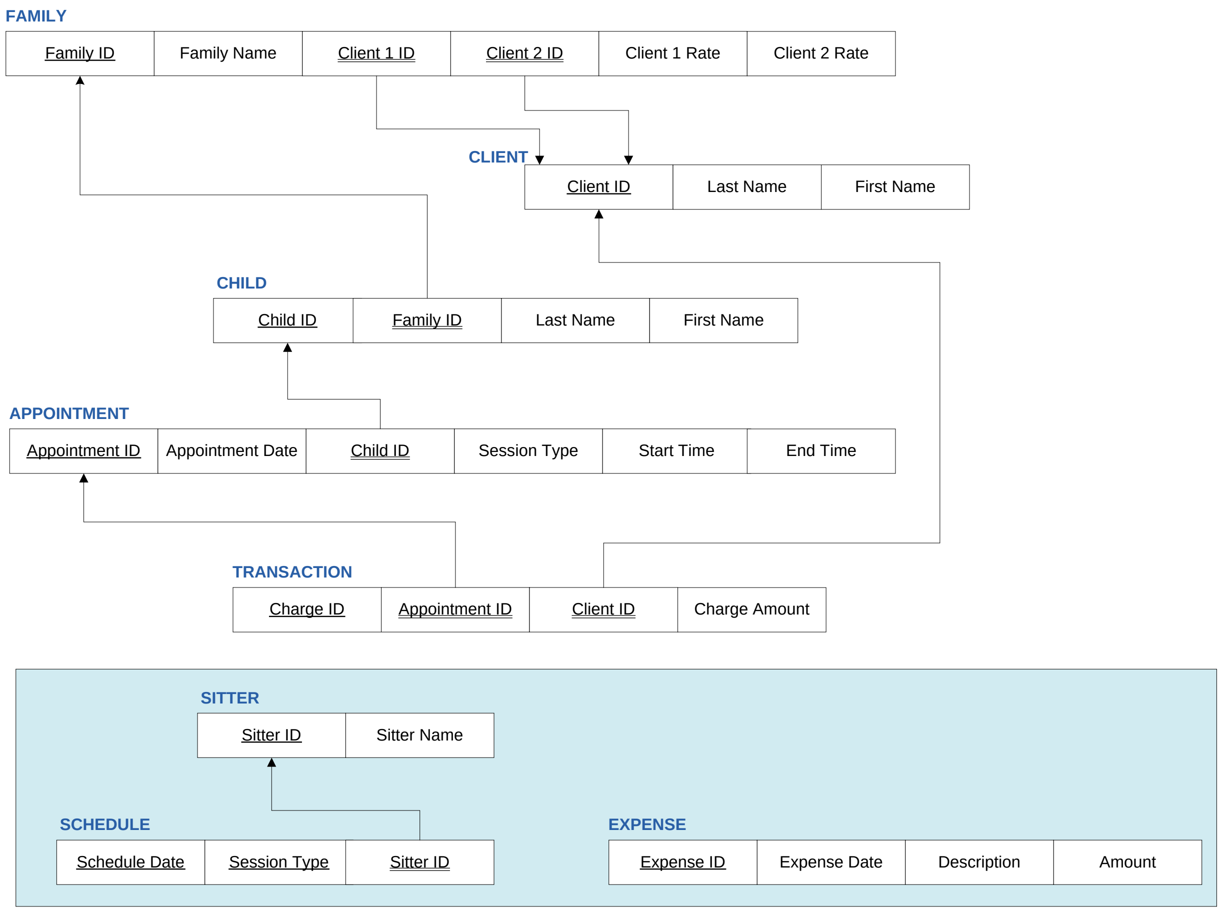
When an Appointment is approved, a Pay record is created for the Sitter who was on the Schedule for the date and time of the Appointment.

Sitter/Schedule

A Schedule exists for each before school slot and each after school slot on a given day. There can therefore be a maximum of two schedules on any given day. Each Schedule record is assigned one Sitter.

Relation Diagram

Note: For readability purposes, not all fields for each table are shown in this diagram. Refer to the Entity-Relationship Diagram for a complete set of fields.



Master File Maintenance

Description and Priority

The application's business functions rely on master files, which include clients, children, family relationships, and sitters. These files must be populated before business functions can be used.

Priority: High

Without master file data, business functions cannot be used.

Stimulus/Response Sequences

Use case name: Add New Child
Actions: • User enters child data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application inserts new child into database.

Use case name: Update Child Data
Actions: • User selects child to update. • User changes child data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates child data.

Use case name: Add New Client
Actions: • User enters client data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application inserts new client into database.

Use case name: Update Client Data
Actions: • User selects client to update. • User changes client data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates client data.

Use case name: Add New Family
Actions: • User enters family data. • User assigns children and clients to family. • User enters base rates for each client, or determines how rate should be split between clients. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application inserts new family into database.

Use case name: Update Family Data
Actions: • User selects family to update. • User changes family data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates family data.

Use case name: Add New Sitter
Actions: • User enters sitter data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application inserts new sitter into database.

Use case name: Update Sitter Data
Actions: • User selects sitter to update. • User changes sitter data. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates sitter data.

Functional Requirements

REQ-1:	Validates input data for acceptable and complete values, notifies user of invalid data, and prevents invalid data from being written to the database.
REQ-2:	Provides the use of default values, where appropriate, to simplify data entry.
REQ-3:	For the family maintenance function, permits children and clients to be assigned to a family only if those children and clients have already been defined. In other words, the user must select from established values and not be allowed to key in a value instead.
REQ-4:	For the family maintenance function, allows charges to be split among two clients either as a percentage or by individual amounts per client.

Application Configuration Maintenance

Description and Priority

Some values are used repeatedly across multiple sessions. Instead of requiring the user to enter these values each time, they can be saved in application configuration settings.

Priority: Low

Application configuration items are a convenience to the user. Business functions can still be used without them, albeit by requiring the user to enter common values.

Stimulus/Response Sequences

Use case name: Maintain Configuration Values
Actions: • User enters or changes one or more configuration items. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates the application configuration items.

Functional Requirements

REQ-1: Validates input data for acceptable and complete values, notifies user of invalid data, and prevents invalid data from being updated to the application configuration file.

Appointment Entry

Description and Priority

Building the transactions that will ultimately be the basis for year-end reports requires the recording of children's schedules, each instance referred to as an appointment. An appointment is created for a given child on a given date for a given appointment type (before school or after school).

Priority: High

Since all reporting is ultimately based on the children's schedules, appointment entry is essential to the application.

Stimulus/Response Sequences

Use case name: Enter New Appointment
Actions: • User selects a child, an appointment type, and one or more dates for which to create appointments. • Application inserts into the database the new appointments, avoiding any duplicates.

Use case name: Update/Delete Appointment
Actions: • User selects an appointment to update or delete. • For updates, the updated appointment data is written to the database. • For deletes, the appointment is deleted from the database.

Use case name: Maintain Sitter Schedule
Actions: • User selects the sitters who will be working on each date, one before school and one after school.

Functional Requirements

REQ-1:	Presents a calendar on which appointments and sitter schedules can be entered.
REQ-2:	Provides a means to change and delete appointments that have already been entered.
REQ-3:	Ensures valid data are entered.

Charge and Payment Entry

Description and Priority

Based on child appointments, amounts due by clients are computed. The user can accept these or modify them as needed. The application also provides for the entry of client payments, as well as adjustments to client accounts.

Priority: High

Charges, payments, and adjustments are a core function of the application.

Stimulus/Response Sequences

Use case name: Validate and Adjust Appointments
Actions: • User validates appointments for which the child was present. • User validates or modifies charge amount for each appointment. • For families with two clients for which pay is not based on a percentage, the user selects the client that is responsible for paying for the appointment. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application creates charge transactions for each validated appointment, and inserts them into the database. • Application marks validated appointments as approved.

Use case name: Payment Entry
Actions: • User selects a client. • User enters payment information, including date and amount. • User applies client balance to outstanding charge transactions. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates transaction table with payment and paid charge information.

Use case name: Adjustment Entry
Actions: • User selects a client. • User enters adjustment information (either debit or credit), including date and amount. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates transaction table with payment and paid charge information.

Functional Requirements

REQ-1:	Presents open appointments for the user to approve.
REQ-2:	Allows user to adjust amount to be charged for a given appointment.
REQ-3:	For two-client families, allows the user to select the client responsible for a given appointment.
REQ-4:	Allows user to select the charges to which a client's outstanding balance should be applied.
REQ-5:	Allows the entry of client payments, including the date and amount of payment.
REQ-6:	Provides a mechanism for the user to enter adjustments to a client's account balance.
REQ-7:	Validates input data, notifying the user of invalid data.

Expense Entry

Description and Priority

To support income tax preparation, expenses related to the child-care business can be recorded in the application.

Priority: Low

Itemized expenses are infrequent with the client's business, so this functionality is the least important of all the application's functions.

Stimulus/Response Sequences

Use case name: Enter Expenses
Actions: • User enters or updates expense items, including date, amount, and description. • Application verifies input. If data is invalid, the user is notified and the use case ends. • Application updates the expense data to the database.

Functional Requirements

REQ-1: Allows user to enter and update expenses.
REQ-2: Validates input data, notifying the user of invalid data.

Reporting

Description and Priority

To support income tax preparation of both the clients and the sitters, year-end reports are to be provided.

Priority: High

These reports are the eventual output of all the data entry functions in the application.

Stimulus/Response Sequences

Use case name: Generate reports
Actions: • User selects a report to generate. • User selects the client or sitter for which the report is to be generated. • Application generates the report.

Functional Requirements

REQ-1: Allows user to select the type of report.
REQ-2: Allows the user to select the client or the sitter for which the report is to be generated.
REQ-3: Generates the selected report.

Section 3 Nonfunctional Requirements

Performance Requirements

None.

Safety Requirements

None.

Security Requirements

Because there are no financial identification items in this project, security is a minor concern. Keeping personal information, like phone numbers, secure will rely on the user keeping the computer secure. The application will have no login function.

Software Quality Attributes

Due to the client's relative lack of experience with computer applications, simplicity is key to the user interface. Applications functions should be easy to learn and made as obvious to use as possible.

To keep the user experience uncomplicated, unnecessary functionality should be avoided.

User errors must be clearly evident.

Because financial information is being recorded, and because much of the information will be used as supporting data for income tax returns, data integrity and accuracy is paramount.

Other Requirements

None.

Chapter 3. Management Plans

Section 1 Work Plan

Task	Completion Date	Description
Learn Key Processes	02/05/2011	This step involves reviewing with the client the content and the methods she uses today to record information and calculate the various figures needed to run the child care service. From there, a list of features needed in the application will be prepared.
Update Knowledge	02/12/2011	Since it has been a while since I have coded a solution in Visual Studio, I will need to familiarize myself with the latest set of Windows form controls and other technology now available.
Design Database	02/20/2011	Based on the process review, the various database tables and their relationships will be designed.
Design User Interface	03/03/2011	The forms and controls needed to support the chosen features will be created.
Create Classes	03/10/2011	The classes for objects used in the application will be coded.
Code Procedures	04/03/2011	The code to support business functions will be coded.
Test	04/11/2011	Functional testing of all features will be performed.
Obtain Approval	04/11/2011	Client approval and feedback will be sought, and the application will be updated per the content of that feedback.
Create Documentation	04/28/2011	The user documentation and materials needed to fulfill Capstone requirements will be produced.

Section 2 Control Plan

The Work Plan will be referred to regularly to compare project progress to estimated target dates. If discrepancies are encountered, they will be handled as follows:

- Those deemed insignificant will result in a modification of the dates in the Work Plan.

- Those deemed significant will be reviewed with the client, resulting in the possible omission of those functions or features deemed to be of low priority.

Section 3 Risk Management Plan

The foremost risk to the project is the full-time work schedule of the developer. This is especially significant because he is currently involved in a major project through which his employer's main information system is being replaced. This has and will lead to increased work hours for several months. If the work schedule interferes with this project's progress, some desired features may have to be omitted.

There is a lesser risk associated with the fact that it has been a few years since the developer has done the type of programming required for this project, and that the Visual Studio product has been updated in the interim. This will require a review of programming concepts, as well as the learning of a newer version of Visual Studio and its new features.

Section 4 Testing Plan (describe how you would do unit testing, regression testing, and integration test. Any testing scenarios you can plan before any code is written)

Unit Testing

Each application function will go through unit testing. Scenarios include:

- Enter new child.
- Update child information.
- Enter new client.
- Update client information.
- Enter new family.
- Update family information.
- Enter a family consisting of a single client.
- Enter a family consisting of two clients, each of which pays a percentage of each appointment's charges.
- Enter a family consisting of two clients, each of which is responsible for selected appointments.
- Enter new sitter.
- Update sitter information.
- Enter/update application configuration settings.
- Enter new appointment.

- Update appointment.
- Delete appointment.
- Enter sitter schedule.
- Change sitter schedule.
- Approve appointment with no changes.
- Approve appointment with charge changes.
- For two-client families who pay based on percentage, change charge amounts to verify percentages.
- For two-client families who do not pay based on percentage, select responsible client for an appointment.
- Apply client balance to charge.
- Enter client payment.
- Enter client credit adjustment.
- Enter client debit adjustment.
- Enter new expense.
- Update expense.
- Generate year-end reports.

Integrated Testing

Integrated testing scenarios include:

- Appointment entry, update, and deletion affects appointment validation function.
- Child entry function updates child lists in other parts of the application.
- Client entry function updates client lists in other parts of the application.
- Application settings affect several parts of the application.
- Charges, payments, and adjustments affect the client balance due.
- Reports reflect accurate snapshots of data for the time period being reported.

Section 5 Project Organization/Decomposition

See Work Plan in Section 1 above.

Appendix: Glossary and References

GLOSSARY

Actual End Time
The time at which an individual appointment actually ended.

Actual Start Time
The time at which an individual appointment actually started.

Appointment
A single care session on a given day—either before school or after school—for a given child.

Appointment Type
The type of appointment, either before school or after school.

Base Rate
The default charge for a given appointment. Base rates can vary by session type and by client, and can be overridden by the user to reflect actual appointment lengths.

Child
An individual who will be cared for in the home before and/or after school.

Client
The customer. An individual who is financially responsible for paying for child care. A client it usually a parent or guardian.

Expense
An item (e.g., supply) used in the business. More specifically, the cost of that item.

Family
A group of one or more children whose care is paid for by one or two clients.

Schedule
A specific session of child care by a sitter. A given schedule is composed of one or more appointments.

Sitter
An individual who provides child care.

Standard End Time
The time at which an individual appointment is scheduled to end.

Standard Start Time
The time at which an individual appointment is scheduled to start.

Transaction
An item that affects a client's balance due. A given transaction can be a charge, a payment, or an adjustment.

REFERENCES

Books referred to for refreshing programming concepts:

Agarwal, Vidya Vrat. *Beginning C# 2008 databases : from novice to professional*. Apress, 2008.

Gross, Christian. *Beginning C# 2008 : from novice to professional*. Apress, 2008.

Kent, Jeffrey A. *Visual C# 2005*. McGraw-Hill, 2006.

Liberty, Jesse. *Programming C# 3.0*. O'Reilly, 2008.

Sphar, Chuck. *C# 2008 for dummies*. Wiley, 2008.

Telles, Matthew A. *C# black book*. Coriolis Group Books, 2002.

Web sites referred to for solving programming issues:

codelog.blogial.com

msdn.microsoft.com

social.msdn.microsoft.com

www.4guysfromrolla.com

www.codeproject.com

www.dotnetperls.com

www.dotnetspider.com

www.experts-exchange.com

www.visualstudiodev.com

Larry Overkamp
Capstone Project
CSC 450   Spring, 2011

Integrated Testing Scenarios

Testing scenarios are presented in the sequence in which the program is normally used.

The Comments column contains a description of the test results. Dark red text signifies an issue that was encountered and had to be corrected in order to run a successful test.

Setup Functions

Scenario	Comments
Add Sitter • Should create a Sitter record in the database. • Should add the sitter name to the sitter listbox on the Setup page. • New sitter should be present in the sitter drop-down lists on the Calendar page. • New sitter should be present in the Sitters listbox on the Payroll page. • New sitter should be present in the sitter listbox on the Taxes page.	The program allowed the user to enter a sitter name that matched one that already existed. This should be prohibited. Logic was inserted to prohibit duplicate names. After correcting the above defect, all expected actions performed successfully.

Scenario	Comments
Change Sitter name *A sitter is selected from the Setup page Sitter listbox. In the sitter name textbox, the name is changed and the OK button pressed.* • Should change the sitter name in the Sitter record in the database. • The changed name should show up in the Setup page sitter listbox.	The program allowed the user to change the sitter name to one that already existed. This should be prohibited. Logic was inserted to prohibit duplicate names when changing a sitter. After correcting the above defect, all expected actions performed successfully.

Scenario	Comments
Cancel Sitter changes *A sitter is selected from the Setup page Sitter listbox. In the sitter name textbox, the name is changed and the Cancel button pressed.* • Should not change the sitter name in the Sitter record in the database. • The name in the Setup page sitter listbox should not have changed.	All expected actions performed successfully.

Scenario	Comments
Delete Sitter, no dependent relationships *If the sitter has no pay records and no schedule records, the deletion should proceed.* • Should delete the Sitter record from the database. • If the sitter was a usual sitter, the usual sitter name should now be "not specified". • Should remove the sitter name from the sitter listbox on the Setup page. • Should remove the sitter from the sitter drop-down lists on the Calendar page. • Should remove the sitter from the Sitters listbox on the Payroll page. • Should remove the sitter from the sitter listbox on the Taxes page.	All expected actions performed successfully.

Scenario	Comments
Attempt to delete Sitter with a Pay record • The program should display a message indicating that the Sitter cannot be deleted. • The Sitter should not be removed from the Sitter table in the database.	All expected actions performed successfully.

Scenario	Comments
Attempt to delete Sitter with a Schedule record • The program should display a message indicating that the Sitter cannot be deleted. • The Sitter should not be removed from the Sitter table in the database.	All expected actions performed successfully.

Scenario	Comments
Check the usual sitter checkboxes for a given sitter • Should update the usual sitter application configuration settings with the sitter's database index number. • Should show the sitter's name as the usual sitter in the Sitters section of the page. • Should be the usual sitter in the drop-down lists on the Calendar page for any date for which a sitter has not already been scheduled.	All expected actions performed successfully.

Scenario	Comments
Uncheck the usual sitter checkboxes for a given sitter • Should update the usual sitter application configuration settings by setting the corresponding setting to an empty string. • Should show "not specified" as the usual sitter in the Sitters section of the tab page. • Should no longer be the usual sitter in the drop-down lists on the Calendar page for any date for which a sitter has not already been scheduled.	All expected actions performed successfully.

Scenario	Comments
Check a usual sitter checkbox for a given sitter that is already checked for another sitter. *If a sitter is designated as the usual sitter for before school, for example, and the usual before school sitter is checked on for another sitter, the usual before school sitter should now be the new sitter.* • Should update the usual sitter application configuration setting to be the number of the new sitter. • The new sitter's name should now be listed as the usual sitter in the Sitters section of the tab page. • The previous usual sitter's checkbox should no longer be checked on the Setup page. • The new sitter should now be the default sitter in the drop-down lists on the Calendar page for any date for which a sitter has not already been scheduled, replacing the previous default sitter.	All expected actions performed successfully.

Scenario	Comments
View configuration settings *The main section of the Setup tab page holds the settings from the application configuration settings file.* • The items in the various setup textboxes should match their corresponding application configuration settings.	All expected actions performed successfully.

Scenario	Comments
Change configuration settings • After changing values, the items in the various setup textboxes should be updated to the application configuration settings file.	An unhandled exception was returned during the first test run. This was encountered with the Approve Future Appointments checkbox, and generated an "Object reference not set to an instance of an object." error. The configuration setting for the checkbox was missing from the configuration file for reasons unknown. Re-inserted it and retested. Subsequent test was successful. All items were updated to the configuration settings file.

Scenario	Comments
Validation: Data types *Each configuration setting except the checkbox is run through validation logic to ensure it is of the appropriate data type (integer, currency, etc.).* • After changing values to invalid values, an error message for the invalid textbox in question should be displayed. • The items should not be updated to the application configuration settings file.	All invalid values were caught. The configuration settings file was not updated with invalid values.

Scenario	Comments
Validation: Time comparisons *The before school and after school start times cannot be earlier than the start limit. Likewise, the before school and after school end times cannot be later than the end limit. This check is performed whether the limit values or the start/end time values are changed.* • The program should display an error message any time the entered times are outside the established limits. • The configuration settings file should not be updated with invalid values.	All invalid values were caught. The configuration settings file was not updated with invalid values. During this testing it was discovered that the end times that were earlier than their respective start times could be entered. Additional validation coding was needed. After additional code was added, all validation logic worked successfully.

Clientele Functions

Scenario	Comments
Initial display • The first and last name of every child in the Child database table should be listed in the children list in the Children section of the page. • The first and last name of every client in the Client database table should be listed in the client list in the Clients section of the page, and in the client list of the Families section of the page. • The name of every family in the Family database table should be listed in the family list in the Families section of the page. • The first and last name of every child in the Child database table that has not been assigned to a family should be listed in the children not in a family list in the Families section of the page.	All expected actions performed successfully.

Scenario	Comments
Add Child *A child is entered by keying in the first and last names, then pressing the OK button. The notes field is optional.* • Should create a Child record in the database. • Should add the child name to the Children list on the Clientele page. • Should add the child name to the list of children not assigned to a family, on the Clientele page. • Should add the child name to the list of children on the Calendar page.	All expected actions performed successfully.

Scenario	Comments
Change Child data *The child whose data is to be changed is selected from the Children list.* Upon selection: • The data for the child should fill the text fields. Upon pressing the OK button: • Should update the Child record in the database. • If the name was changed, the new name should be displayed: o In the Children list on the Clientele page. o In the Children not yet in a family list on the Clientele page, if the child was listed there prior to the change. o In the list of children on the Calendar page. o For any appointments for that child on the Calendar page. o In the Ledger list. o In the transaction history list on the Billing page. o In the transaction history list on the Payroll page.	When the OK button was pressed without changing the name, the program displayed an error message stating that a child with that name already existed. That error message should be displayed only if the user is entering a new child, or if changing a child name to an existing child name, not if the name hasn't changed. Also, the changed name did not show up in the appointments on the calendar page. (The name change was reflected in all other places listed at left.) After correcting the above defects, all expected actions performed successfully.

Scenario	Comments
Delete Child *A child can be deleted only if there are no Appointment records for the child. (That validation is covered by a separate scenario in this section.)* • Should delete the Child record from the database. • Should remove the child name from the Children list on the Clientele page. • Should remove the child name from the list on the Clientele page of children not assigned to a family. • Should remove the child name from the list of children on the Calendar page.	All expected actions performed successfully.

Scenario	Comments
Validation: Both names required • Should display an error message if either the first name or the last name has been omitted.	All expected actions performed successfully.

Scenario	Comments
Validation: Delete Child who has Appointments *A child can be deleted only if there are no Appointment records for the child.* • Should display a message stating that the child cannot be deleted. • Should not delete the child from the database.	All expected actions performed successfully.

Scenario	Comments
Validation: Duplicate child name *No two children can have the same name.* If a child is added with, or an existing name is changed to, a name that already exists: • Should display a message stating that the name is a duplicate. • Should not add or update the Child record to the database.	All expected actions performed successfully.

Scenario	Comments
Add Client *A client is entered by keying in the first and last names, then pressing the OK button. The other client fields are optional.* • Should create a Client record in the database. • Should add the client name to the Clients list in the Clients section of the Clientele page. • Should add the client name to the Clients list in the Families section of the Clientele page. • The new name should be displayed in the list of clients on the Billing page. • The new name should be displayed in the list of clients on the Taxes page.	All expected actions performed successfully.

Scenario	Comments
Change Client data *The client whose data is to be changed is selected from the Clients list.* Upon selection: • The data for the client should fill the text fields. Upon pressing the OK button: • Should update the Client record in the database. • If the name was changed, the new name should be displayed in the Clients list in the Clients section of the Clientele page. • If the name was changed, the new name should be displayed in the Clients list in the Families section of the Clientele page. • If the name was changed, the new name should be displayed on the ledger lines on the Ledger page. • If the name was changed, the new name should be displayed in the list of clients on the Billing page. • If the name was changed, the new name should be displayed in the list of clients on the Taxes page.	When the OK button was pressed without changing the name, the program displayed an error message stating that a client with that name already existed. That error message should be displayed only if the user is entering a new client, or if changing a client name to an existing client name, not changing an existing client. After correcting the above defect, all expected actions performed successfully.

Scenario	Comments
Delete Client *A client can be deleted only if there are no Transaction records for the client. (That validation is covered by a separate scenario in this section.)* • Should delete the Client record from the database. • Should remove the client name from the Clients list in the Clients section of the Clientele page. • Should remove the client name from the Clients list in the Families section of the Clientele page. • Should remove the client name from the list of clients on the Billing page. • Should remove the client name from the list of clients on the Taxes page.	All expected actions performed successfully.

Scenario	Comments
Validation: Both names required • Should display an error message if either the first name or the last name has been omitted.	All expected actions performed successfully.

Scenario	Comments
Validation: Phone numbers • Phone numbers must pass the common phone number validation routine. • The Client record must not be updated with invalid phone numbers.	The text of the error message for the cell phone was inaccurate: It stated “Home” instead of “Cell”. Changed the text of the error message. Invalid values in all the phone number fields were caught. The database was not updated with invalid phone numbers.

Scenario	Comments
Validation: Delete Client who has Transactions *A client can be deleted only if there are no Transaction records for the client.* • Should display a message stating that the client cannot be deleted. • Should not delete the client from the database.	All expected actions performed successfully.

Scenario	Comments
Validation: Duplicate client name *No two clients can have the same name.* If a client is added with, or an existing name is changed to, a name that already exists: • Should display a message stating that the name is a duplicate. • Should not add or update the Client record to the database.	All expected actions performed successfully.

Scenario	Comments
Add Family *The only required field for adding a new family is the family name. The other fields are optional.* • Should create a Family record in the database. • Should add the family name to the Families list on the Clientele page.	The family was added successfully.

Scenario	Comments
Delete Family *A family can be deleted only if there are no children assigned to that family. (That validation is covered by a separate scenario in this section.)* • Should delete the Family record from the database. • Should remove the family name from the list of families in the Families section of the Clientele page.	All expected actions performed successfully.

Scenario	Comments
Add child to family *A child is added to a family by clicking on the child name in the “Children not yet assigned to a family” list, then clicking the left arrow button to the left of that list. Multiple children can be added with one button press by clicking on multiple child names.* • Should add the child name to the “Children in this family” list. • Should remove the child name from the “Children not yet assigned to a family” list. If the OK button is pressed: • The family number of the family currently being maintained should be inserted into the Child’s database record.	The child name was removed from the no family list and added to the in family list. The Child records were updated with the family number. The tests were run both for a single child selection and when multiple children were selected.

Scenario	Comments
Remove child from family *A child is removed from a family by clicking on the child name in the "Children in this family" list, then clicking the right arrow button to the right of that list. Multiple children can be removed with one button press by clicking on multiple child names.* • Should remove the child name from the "Children in this family" list. • Should add the child name to the "Children not yet assigned to a family" list. If the OK button is pressed: • The family number in the Child database record should be set to 0 (zero).	The child name was removed from the in family list and added to the no family list. The Child records were updated with a family number of zero. The tests were run both for a single child selection and when multiple children were selected.

Scenario	Comments
Add client to family *A client is added to a family by clicking on the client name in the Clients list of the Families section of the Clientele page, then clicking on one of the right arrow buttons to the right of that list. A First Client must be added before a Second Client can be added.* • Should place the name of the client next to the First Client label or the Second client label, depending on the button that was pressed. If the OK button is pressed: • The client number of the First Client should be updated to the Family database record. • If a Second Client was selected, the client number should be updated to the Family database record.	During the process of testing this feature, it was noticed that sometimes the second client base rate textboxes would be enabled when a family had only one client. It was found that this had to do with when the percentage checkbox state had changed. Logic was added to enable them only if there was a second client. After correction of the above defect… Client selections were displayed appropriately. Client numbers were added to the Family database record.

Scenario	Comments
Change a family's client(s) *A family's clients are changed by using the same process as adding a client.* • Should place the name of the client next to the First Client label or the Second client label, depending on the button that was pressed. If the OK button is pressed: • The client number of each selected client should be updated to the Family database record.	Client selections were displayed appropriately. Client numbers were updated in the Family database record.

Scenario	Comments
Delete a second client *The program allows only the second client to be deleted because each family should have at least one client. The second client is deleted by clicking on the Delete Second Client button.* • Should remove the name of the second client from next to the Second Client label. • Should clear and disable the second client's base rate fields, along with the associated default buttons. If the OK button is pressed: • The second client number of the Family database record being maintained should be set to zero.	The second client name was removed. The second client base rate fields were not cleared out and were not disabled, and the associated buttons that insert the default rates were not disabled. After the above defect was corrected, the base rate fields and associated buttons were disabled as expected. The second client number in the Family database record was set to zero.

Scenario	Comments
Default base rate buttons *The base rate fields for each client can be set to the default value (from the Setup page) by clicking on the "checkmark" buttons next to them. There are four such buttons: Two for each client, one for the before school rate and the other for the after school rate.* • Each button should place the appropriate base rate default in its corresponding base rate text field.	All buttons worked as expected.

Scenario	Comments
Change family data *The various controls that are used to configure the family are filled in when a family is selected from the family list.* • Each child with the family's family number should be displayed in the "Children in this family" list. • Any notes recorded for the family should be shown in the Notes field. • Any clients assigned to the family should be listed in the First Client and Second Client spaces. • If there is one client, that client's before school and after school base rates should be displayed, and the controls pertaining to percentages should be empty and disabled. • If there are two clients, and they split the cost by percentage, then the percentage checkbox should be checked, the percent figures should be displayed, and the before school and after school base rates for the first client should be filled in. • If there are two clients, and they do not split the cost by percentage, then the percentage checkbox should not be checked, the percent values should be zero, and the base rates for both clients should be displayed.	In each case, the appropriate controls were filled in with the appropriate data from the Family database record.

Scenario	Comments
Percentage usage • If the family has two clients, the percentage checkbox should be enabled; if the family has one client, the checkbox should be disabled. • If a family has two clients, and the second client is removed from the family, the percentage checkbox should be disabled. • If the percentage checkbox is checked, the percentage textbox should be enabled, and the second client's base rate fields should be set to zero and should be disabled, and the default buttons associated with the second client's base rate fields should be disabled. • If the percentage checkbox is unchecked, the percentage textbox should be disabled, and the second client's base rate fields should be enabled.	The default buttons for the second client's base rate fields were not disabled. Once coding was inserted to address the above defect, all expected outcomes were achieved.

Scenario	Comments
Validation: Duplicate family name *No two families can have the same name.* If a family is added with, or an existing name is changed to, a name that already exists: • Should display a message stating that the name is a duplicate. • Should not add or update the Client record to the database.	Duplicate names were caught when adding a name but not when changing a name. A family name could be changed to one that already exists. Modified the code to correct this defect. Duplicate names were caught in both cases.

Scenario	Comments
Validation: Delete Family with assigned child *A family cannot be deleted if there are any children assigned to the family.* If an attempt is made to delete a family that has at least one child assigned to it: • Should display a message stating that the family cannot be deleted, and the reason for the prohibition. • Should not delete the Family record from the database.	The error message was displayed. The family was not deleted from the database.

Scenario	Comments
Validation: Duplicate clients assigned to family *If a family has two clients, they cannot both be the same.* If both clients are the same, when the user clicks one of the buttons for assigning a client to a family: • Should display an error message. (This should occur whether it is the first client that is being added or the second client is being added.)	Duplicates were detected for both client add buttons.

Scenario	Comments
Validation: Percentage value • Should display an error message if the percentage value is not a one- or two-digit value.	The error message was displayed.

Scenario	Comments
Validation: Client Base Rates • The four base rate fields must be valid decimal values. If any is invalid, an error message should be displayed.	An invalid value was caught for each of the base rate fields.

Calendar Functions

Scenario	Comments
Current date highlighted • The background color of the current date's cell should be blue.	The current date was highlighted appropriately.

Scenario	Comments
Approved appointments highlighted • Each appointment on the calendar that has already been approved should be highlighted (the row should have a background color) to signal to the user that it cannot be changed or deleted.	All such appointments were highlighted appropriately.

Scenario	Comments
Calendar navigation: Previous Week • The period for which the calendar is displayed should be one week earlier than the start of the currently-displayed period. • All appointments and sitter schedules should reflect those of the new set of dates shown. • Any dates for which a sitter schedule has not yet been created should show the usual before school and after school sitters, if either has been selected on the Setup page.	All expected actions performed successfully.

Scenario	Comments
Calendar navigation: Next Week • The period for which the calendar is displayed should be one week after the start of the currently-displayed period. • All appointments and sitter schedules should reflect those of the new set of dates shown. • Any dates for which a sitter schedule has not yet been created should show the usual before school and after school sitters, if either has been selected on the Setup page.	All expected actions performed successfully.

Scenario	Comments
Calendar navigation: Current Week • The period for which the calendar is displayed should be the three weeks beginning with the week prior to the current week, making the current week the center row of the calendar. • All appointments and sitter schedules should reflect those of the new set of dates shown. • Any dates for which a sitter schedule has not yet been created should show the usual before school and after school sitters, if either has been selected on the Setup page. • If the current week is already in the center of the calendar, the program should recognize this and not bother rebuilding the calendar.	All expected actions performed successfully.

Scenario	Comments
Appointment Entry: One child, before school *For this scenario, one child is selected, the Before School appointment type is selected, and one or more days are selected. Appointments are created after those selections have been made and the user has pressed the Save button.* • During the selection process, the numbers of the dates that the user has selected should be highlighted; those deselected should no longer be highlighted. • After a child, an appointment type, and at least one date have been selected, the calendar navigation buttons should be disabled so the user can't navigate away without handling the current selections. • After the Save button has been pressed, an appointment row is inserted in the Appointment table for each selected date. • After the Save button has been pressed, a schedule record should be created in the Schedule table for the before school sitter if a before school sitter has been selected and if such a record does not already exist in the Schedule database. • After the Save button has been pressed, the child's name and the default before school start and end times should be displayed—in the before school text color—on the selected dates. • After the Save button has been pressed, the selection in the child list should be cleared, the appointment type radio buttons should be cleared, all selected dates should no longer be highlighted, and the calendar navigation buttons should be enabled.	All expected actions performed successfully.

Scenario	Comments
Appointment Entry: One child, after school *For this scenario, one child is selected, the After School appointment type is selected, and one or more days are selected. Appointments are created after those selections have been made and the user has pressed the Save button.* • The characteristics of this scenario are the same as the before school scenario, applied to the after school appointment type. The appointment entry shown on the calendar should be in the after school color, and the start and end times should be the default after school start and end times.	All expected actions performed successfully.

Scenario	Comments
Appointment Entry: One child, both before school and after school *For this scenario, one child is selected, the Both appointment type is selected, and one or more days are selected. Appointments are created after those selections have been made and the user has pressed the Save button.* • The characteristics of this scenario are the same as the before school and after school scenarios, applied to both the before school and the after school appointment type. The appointment entries shown on the calendar should be in the appropriate colors, and the start and end times should be the defaults.	All expected actions performed successfully.

Scenario	Comments
Appointment Entry: Two children, both before school and after school *For this scenario, two children are selected, the Both appointment type is selected, and one or more days are selected. Appointments are created after those selections have been made and the user has pressed the Save button.* • The characteristics of this scenario are the same as previous appointment entry scenarios, but appointments are created for each selected child.	All expected actions performed successfully.

Scenario	Comments
Cancel Appointment Entry *For this scenario, at least one child has been selected, an appointment type has been selected, and at least one date has been selected. The Cancel button is then pressed.* • The selected items in the child list should no longer be selected. • No appointment type radio button should be selected. • All selected dates should no longer be highlighted. • The calendar navigation buttons should be enabled. • No appointments should be inserted into the Appointment database table.	All expected actions performed successfully. No appointments were inserted into the Appointment database table.

Scenario	Comments
Change Sitter: Select sitter where previously there was not one *No Schedule records exist for a given date and appointment type combination until either an appointment is made for that combination, or the user changes the default value displayed in that combination's drop-down box. This scenario changes the sitter for a date/type combination where one did not exist before (dashes are shown in the drop-down list).* • An entry in the Schedule table should be made for the date/type, with the selected sitter's number.	The appropriate entry was made in the Schedule table.

Scenario	Comments
Change Sitter: Select a sitter that is different from the previous one *Change the sitter from one name to another. Neither the original or new selection is the dashed line item.* • The entry in the Schedule table for the date/type in question should be changed, with the previous sitter number being changed to the newly-selected sitter's number.	The appropriate entry was updated in the Schedule table.

Scenario	Comments
Change Sitter: Change sitter to no selection *Change the drop-down list from a sitter name to the dashes entry.* • The entry in the Schedule table for the date/appointment type combination should be deleted.	The appropriate entry was deleted from the Schedule table.

Scenario	Comments
Click on Appointment: Approved appointment *An appointment that has been approved is highlighted on the calendar, and cannot be modified or deleted.* • A dialog box that explains that the appointment cannot be changed or deleted should be displayed.	The dialog box was displayed.

Scenario	Comments
Click on Appointment: Appointment that has not been approved *When a non-approved appointment is clicked on, a sub-form is displayed on which the user can change the appointment times or can delete the appointment.* • The sub-form should be displayed.	The sub-form was displayed.

Scenario	Comments
Delete Appointment *This is performed via the sub-form that is displayed when the user clicks on a non-approved appointment. The user clicks on the Delete Appointment button.* • The appointment should be deleted from the database. • The appointment should no longer show up on the calendar.	All expected actions performed successfully.

Scenario	Comments
Change appointment time using time change buttons *The buttons to the left and right of the time fields make the times earlier and later, respectively, by the increment specified on the Setup page.* • The left button for both time fields should make the time earlier by the number of minutes specified in the Setup increment field. • The right button for both time fields should make the time later by the number of minutes specified in the Setup increment field. • The time should not be allowed to be before the start limit, nor later than the end limit, specified on the Setup page. • When the OK button is pressed, the appointment time should be reflected on the Calendar.	All expected actions performed successfully.

Scenario	Comments
Change appointment time by keying in the times • The time should not be allowed to be before the start limit, nor later than the end limit, specified on the Setup page. • When the OK button is pressed, the appointment time should be reflected on the Calendar.	All expected actions performed successfully.

Scenario	Comments
Cancel a changed appointment When the user clicks the Cancel button on the appointment sub-form after changing the time(s): • The sub-form should be closed. • The appointment time should be what it was originally when the appointment was clicked (i.e., it should not have changed).	All expected actions performed successfully.

Scenario	Comments
Validation: Appointment times • The time fields are validated for correct format. An invalid time should trigger a dialog box with an error message. • The appointment should not be updated with changed data.	All expected actions performed successfully.

Ledger Functions

Scenario	Comments
Initial display • The ledger table should display up to 17 appointments that have not yet been approved. Each line should have the appointment date, start and end times, and child name. Each line should also contain the client(s) responsible for payment, the payment amount for the appointment, and the name of the sitter on duty at that time. • If the family to which the child belongs has a single client, only that client should be present on the line. • If the family to which the child belongs has two clients who pay by percentage, both clients should be displayed, along with their calculated percentage of the charge. • If the family to which the child belongs has two clients who do not pay by percentage, both clients should be displayed, along with radio buttons for selecting the client responsible for the appointment. • If every data item is present on a given line, a checkbox is also displayed. This checkbox is used for approving the appointment. Data items that must be present include client(s) and sitter. • If a given appointment's start and/or end date is different from the default, the program calculates the difference based on the increment charge entered on the setup page.	All expected actions performed successfully.

Scenario	Comments
Future appointments *A Setup page configuration checkbox is used to indicate whether appointments that begin after the current date/time should be displayed on the Ledger page.* • When the future appointments checkbox is checked, appointments that start after the current date/time should show up in the Ledger. • When the future appointments checkbox is not checked, appointments that start after the current date/time should not show up in the Ledger.	Appointments with start times after the current date/time showed up only when the future appointments checkbox on the Setup page was checked.

Scenario	Comments
Items check/uncheck • When the Check All Items button is pressed, all checkboxes on the ledger should be checked. • When the Uncheck All Items button is pressed, all checkboxes on the ledger should be unchecked.	The buttons worked as expected.

Scenario	Comments
Validation: Error indicators • Each ledger line that has an invalid item should have a red asterisk at its left end, and the generic error message should appear at the right side of the window.	The error indicators worked as expected.

Scenario	Comments
Validation: Charge amounts • Each charge amount on each ledger row for which the checkmark is checked is validated through the common currency validation routine. • No appointments with an invalid charge amount should be accepted.	Invalid charge amounts were flagged. No appointments with invalid charges were accepted.

Scenario	Comments
Validation: Selectable clients *For those families who do not pay on a percentage basis, radio buttons are displayed for the user to select the responsible client.* • A ledger row should be flagged as in error if neither radio button has been checked.	Rows with radio buttons where neither button was checked were flagged.

Scenario	Comments
Valid Ledger Item: One client *In this scenario, the ledger line has one client. It is assumed that all validation has been passed.* • The Approved flag in the appointment's record in the Appointment database table should be set to true. • A Transaction record should be created for the client, with data in all columns pertinent to a charge entry. • A Pay record should be created for the sitter, with data in all columns pertinent to a charge entry. • The appointment should no longer show up in the ledger.	Although a Transaction record and a Pay record were created, neither contained the appointment number. The code was reviewed, and the problem was identified and fixed. All database updates and inserts were successful. The appointment no longer showed up in the ledger.

Scenario	Comments
Valid Ledger Item: Two clients, percentage-based *In this scenario, the ledger line has two clients, and they each pay a percentage of the total charge. It is assumed that all validation has been passed.* Upon ledger display: • The charge displayed for each client should reflect that client's percentage of the total. Upon accepting checked items: • The Approved flag in the appointment's record in the Appointment database table should be set to true. • A Transaction record should be created for each client, with data in all columns pertinent to a charge entry. The transaction amount in each record should reflect the client's percentage of the total. • A Pay record should be created for the sitter, with data in all columns pertinent to a charge entry. The pay amount should be the total for the appointment (i.e., the sum of the two clients' charges). • The appointment should no longer show up in the ledger.	The percentages displayed reflected the percentage defined for the family. All database updates and inserts were successful. The appointment no longer showed up in the ledger.

Scenario	Comments
Valid Ledger Item: Two clients, not percentage-based *In this scenario, the ledger line has two clients, and they don't pay based on a percentage. Instead, only one of the clients is responsible for the appointment. It is assumed that all validation has been passed.* Upon ledger display: • The identical charge amount is displayed for both clients. • Each client name is preceded by a radio button for selecting the responsible client. Upon radio button selection: • The charge amount text field for the client who was not selected should be disabled. Upon accepting checked items: • The Approved flag in the appointment's record in the Appointment database table should be set to true. • A Transaction record should be created for the selected client, with data in all columns pertinent to a charge entry. • A Pay record should be created for the sitter, with data in all columns pertinent to a charge entry. The pay amount should be the amount specified for the selected client. • The appointment should no longer show up in the ledger.	The displayed charge amounts were correct. The radio buttons were displayed. Selecting one client's radio button disabled the charge amount text field for the other client. All database updates and inserts were successful. The appointment no longer showed up in the ledger.

Billing Functions

Scenario	Comments
Select Client • Should display the client's unpaid balance. • A credit balance should be displayed in red. • Should display in the transaction history list all the client's transactions that are in the Transaction database table. • All entries in the history list should list the date, type, and amount of the transaction. • Payment amounts in the history list should be negative values. • A charge entry in the history list should show the appointment type (before or after school) and the child name. • A payment entry in the history list should show the check number (or "Cash") if one was entered at time of payment entry. • An adjustment entry in the history list should show the adjustment description. • Should enable all payment entry controls and all adjustment entry controls.	All expected actions performed successfully.

Scenario	Comments
Enter Payment Upon clicking the OK button: • Should insert the payment into the Transaction database table. • Should update the unpaid balance shown on the page. • Should add the payment to the transaction history list. • Should clear the payment entry textboxes.	All expected actions performed successfully.

Scenario	Comments
Enter Adjustment (debit) Upon clicking the OK button: • Should insert the adjustment into the Transaction database table. • Should update the unpaid balance shown on the page, increased by the amount of the adjustment. • Should add the adjustment to the transaction history list, showing a positive value. • Should clear the adjustment entry textboxes.	All expected actions performed successfully.

Scenario	Comments
Enter Adjustment (credit) Upon clicking the OK button: • Should insert the adjustment into the Transaction database table. • Should update the unpaid balance shown on the page, decreased by the amount of the adjustment. • Should add the adjustment to the transaction history list, showing a negative value. • Should clear the adjustment entry textboxes.	All expected actions performed successfully.

Scenario	Comments
Cancel Payment Upon clicking the Cancel button: • All payment textboxes should be cleared.	All payment textboxes were cleared.

Scenario	Comments
Cancel Adjustment Upon clicking the Cancel button: • All adjustment textboxes should be cleared.	All adjustment textboxes were cleared.

Scenario	Comments
Use the Cash button In the payment entry section of the page, the Cash button: • Should insert the word Cash into the check number textbox.	Success.

Scenario	Comments
Validation: Enter future date *Future dates are permitted for both the payment date and the adjustment date. However the program warns the user that such a date was entered.* • If a payment date that is after the current date is entered, the program should warn the user and ask if the entry of the future date was intentional. • If an adjustment date that is after the current date is entered, the program should warn the user and ask if the entry of the future date was intentional.	Future date entries for both date fields were caught, and the warning displayed.

Scenario	Comments
Validation: Payment data • The payment date must be a valid date in m/d/yyyy format. • The payment amount must be a valid decimal-type number. • The payment amount must be greater than zero. • A payment with invalid data must not be written to the Transaction table. • No validation is done on the check number field.	All invalid values were caught. The Transaction table was not updated with invalid values.

Scenario	Comments
Validation: Adjustment data • The adjustment date must be a valid date in m/d/yyyy format. • The adjustment amount must be a valid decimal-type number. (Negative adjustments are allowed.) • The adjustment description must be entered. • An adjustment with invalid data must not be written to the Transaction table.	All invalid values were caught. The Transaction table was not updated with invalid values.

Payroll Functions

Scenario	Comments
Select Sitter • Should display the amount due the sitter. • A credit balance should be displayed in red. • Should display in the transaction history list all the sitter's transactions that are in the Pay database table. • All entries in the history list should list the date, type, and amount of the transaction. • Payment amounts in the history list should be negative values. • A charge entry in the history list should show the appointment type (before or after school) and the child name. • A payment entry in the history list should show the check number (or "Cash") if one was entered at time of payment entry. • An adjustment entry in the history list should show the adjustment description. • Should enable all payment entry controls and all adjustment entry controls.	All expected actions performed successfully.

Scenario	Comments
Enter Payment Upon clicking the OK button: • Should insert the payment into the Pay database table. • Should update the pay due amount shown on the page. • Should add the payment to the transaction history list. • Should clear the payment entry textboxes.	All expected actions performed successfully.

Scenario	Comments
Enter Adjustment (debit) Upon clicking the OK button: • Should insert the adjustment into the Pay database table. • Should update the pay due amount shown on the page, increased by the amount of the adjustment. • Should add the adjustment to the transaction history list, showing a positive value. • Should clear the adjustment entry textboxes.	All expected actions performed successfully.

Scenario	Comments
Enter Adjustment (credit) Upon clicking the OK button: • Should insert the adjustment into the Pay database table. • Should update the pay due amount shown on the page, decreased by the amount of the adjustment. • Should add the adjustment to the transaction history list, showing a negative value. • Should clear the adjustment entry textboxes.	All expected actions performed successfully.

Scenario	Comments
Cancel Payment Upon clicking the Cancel button: • All payment textboxes should be cleared.	All payment textboxes were cleared.

Scenario	Comments
Cancel Adjustment Upon clicking the Cancel button: • All adjustment textboxes should be cleared.	All adjustment textboxes were cleared.

Scenario	Comments
Use the Cash button In the payment entry section of the page, the Cash button: • Should insert the word Cash into the check number textbox.	Success.

Scenario	Comments
Validation: Enter future date *Future dates are permitted for both the payment date and the adjustment date. However the program warns the user that such a date was entered.* • If a payment date that is after the current date is entered, the program should warn the user and ask if the entry of the future date was intentional. • If an adjustment date that is after the current date is entered, the program should warn the user and ask if the entry of the future date was intentional.	Future date entries for both date fields were caught, and the warning displayed.

Scenario	Comments
Validation: Payment data • The payment date must be a valid date in m/d/yyyy format. • The payment amount must be a valid decimal-type number. • The payment amount must be greater than zero. • A payment with invalid data must not be written to the Transaction table. • No validation is done on the check number field.	All invalid values were caught. The Pay table was not updated with invalid values.

Scenario	Comments
Validation: Adjustment data • The adjustment date must be a valid date in m/d/yyyy format. • The adjustment amount must be a valid decimal-type number. (Negative adjustments are allowed.) • The adjustment description must be entered. • An adjustment with invalid data must not be written to the Transaction table.	All invalid values were caught. The Pay table was not updated with invalid values.

Taxes Functions

Scenario	Comments
Initial Display *The information initially displayed on the Taxes tab is for the current year.* • Each client should be displayed in the Clients list, along with the client's total payments for the year. • Each sitter should be displayed in the Sitters list, along with the sitter's total payments for the year. • The page should display the number of days and the number of hours that the day care was run during the year. Hours are computed by adding to the total the longest appointment in a given time slot (before school or after school) in a given date. • The date, amount, and description of each expense entered during the year should be displayed in the Expenses list.	All expected actions performed successfully.

Scenario	Comments
Change reporting period *There are buttons on the screen for showing different time periods, as well as a textbox that allows the user to enter a specific year.* • The description at the top of the page should reflect the time period of the button that was used. • The client, sitter, hours used, and expense data displayed should reflect the time period of the button that was used.	The information was updated correctly for each time period selected.

Scenario	Comments
Enter an Expense Upon clicking the OK button: • Should insert the expense into the Expense database table. • Should add the expense to the list of expenses on the screen. • Should clear the expense entry textboxes, and disable the expense entry buttons.	All expected actions performed successfully.

Scenario	Comments
Change an Expense *When an expense in the list is clicked on, its values are placed in the expense entry textboxes.* Upon selecting an expense: • Should place the values in the textboxes. • Should enable the OK, Cancel, and Delete buttons. *The expense entries can then be changed.* Upon clicking the OK button: • Should update the new expense data to the database. • Should show the new data in the list of expenses on the screen. • Should clear the expense entry textboxes, and disable the expense entry buttons.	All expected actions performed successfully.

Scenario	Comments
Delete an expense *To delete an expense, it has to be selected from the list.* Upon clicking the Delete button: • The expense should be removed from the database. • The expense should no longer appear in the list of expenses on the screen. • The expense textboxes should be cleared, and the expense entry buttons should be disabled.	All expected actions performed successfully.

Scenario	Comments
Validation: Selected year *This validation is performed on the value in the selected year textbox when the user has pressed the corresponding button.* • The value must be a valid integer that is not more than four digits long.	All expected actions performed successfully.

Scenario	Comments
Validation: Expense data *This validation is performed on the values in the expense entry section of the page, after the user has pressed the OK button.* • The expense date must be a valid date in m/d/yyyy format. • The expense amount must be a valid decimal-type number. • The description field must not be empty. • An expense with invalid data must not be written to the Expense database table.	All invalid values were caught. The Expense database table was not updated with invalid values.

Common Functions

The functionality in this section is common to more than one application tab page.

Scenario	Comments
Reformatting Function: Dates *As a convenience to the user, some shortcuts can be taken when entering date values, which will, if the entered value is a valid basis for a date value, be reformatted into a complete date value when focus leaves the textbox.* *If the user keys in a t or a T, the current date will be inserted.* *If the user keys in a y or a Y, yesterday's date will be inserted.* *If the user enters a t or a T, followed by a – or a + and one to three digits, the date that the operation represents is displayed. For example t-2 will display the date that was two days ago.* *If the user enters a month and day, separated by a slash, the current year will be appended to the value.* Fields on various tab pages are run through this logic. Date fields on Billing page: • Payment Date • Adjustment Date Date fields on Payroll page: • Payment Date • Adjustment Date	All Billing page date fields were reformatted appropriately. All Payroll page date fields were reformatted appropriately.

Scenario	Comments
Reformatting Function: Times *As a convenience to the user, some shortcuts can be taken when entering time values, which will, if the entered value is a valid basis for a time value, be reformatted into a complete time value when focus leaves the textbox.* *If an entered value ends with an upper- or lowercase A or P, that part of the value will be modified to AM or PM, respectively.* *If the user omits the AM/PM part of the value, and the time field represents a before school time, the program will insert the AM. For an after school field, the program will insert a PM.* Fields on various tab pages are run through this logic. Time fields on Setup page: • Start Limit • End Limit • Before School Start Time • After School Start Time • Before School End Time • After School End Time Time fields on the appointment sub-form of the Calendar page: • Start Time • End Time	All Setup page time fields were reformatted appropriately. All Calendar sub-form time fields were reformatted appropriately.

Scenario	Comments
Reformatting Function: Currency *As a convenience to the user, some shortcuts can be taken when entering currency values, which will, if the entered value is a valid basis for a currency value, be reformatted into a complete currency value when focus leaves the textbox.* *Examples:* *If the user enters a single digit such as 5, the program will reformat it to 5.00.* *If the user enters a value like 5.5, it will be reformatted to 5.50.* Fields on various tab pages are run through this logic. Currency fields on Setup page: • Increment Charge • Before School Default Charge • After School Default Charge Currency fields on the Clientele page: • First Client Before School Base Rate • First Client After School Base Rate • Second Client Before School Base Rate • Second Client After School Base Rate Currency fields on the Ledger page: • Client charge fields Currency fields on Billing page: • Payment Amount • Adjustment Amount Currency fields on Payroll page: • Payment Amount • Adjustment Amount	All Setup page currency fields were reformatted appropriately. The Base Rate fields on the Clientele page did not get reformatted. Reviewing the code showed that no event handlers for this purpose had been defined. After the event handlers for the above defect were inserted, all Clientele currency fields were reformatted appropriately. All Ledger page currency fields were reformatted appropriately. All Billing page currency fields were reformatted appropriately. All Payroll page currency fields were reformatted appropriately.

Scenario	Comments
Reformatting Function: Names *As a convenience to the user, textboxes into which names are entered are reformatted when focus leaves the textbox. The reformatting consists of making the first character uppercase and the rest of the characters lowercase.* Fields on various tab pages may be run through this logic. Name fields on Clientele page: • Child First Name • Child Last Name • Client First Name • Client Last Name • Family Name	All Clientele page child name fields were reformatted appropriately. All Clientele page client name fields were reformatted appropriately. The Clientele page family name was reformatted appropriately.

Scenario	Comments
Reformatting Function: Phone Numbers *As a convenience to the user, textboxes into which phone numbers are entered are reformatted when focus leaves the textbox. The reformatting consists of stripping all non-digit characters from the value, then—if the value is 7, 10, or 11 characters in length—inserting hyphens in the appropriate places.* Fields on various tab pages may be run through this logic. Phone Number fields on Clientele page: • Client Home Phone • Client Work Phone • Client Cell Phone	All Clientele page client phone numbers were reformatted appropriately.

Larry Overkamp
Capstone Project
CSC 450   Spring, 2011

Code Highlights

Due to code statement length and indentation, the text in this section is presented in landscape orientation to improve readability.

Contents

Data Entry Shortcuts

As a convenience to the user, the data entry fields for some common data types use event handlers to reformat abbreviated values into properly-formatted, expanded values. For example, although dates are comprised of month, day, and year—with slash separators—the date field shortcut reduces the number of characters that have to be entered in some circumstances. These event handlers are activated upon leaving the control into which they are entered (by using the Tab key, for example, or clicking on a button). The code for two of the data entry shortcuts is presented in this section.

Date Shortcut

This method takes a string as a parameter and, if it recognizes it as a date shortcut, reformats it into a complete date value and returns it. All returned date values include month, day, and year with century, separated by slashes (e.g., 4/28/2011). A description of these shortcuts is included in the comments at the beginning of the method.

```
public string CommonDateTextLeaveHandler(string parText) {
/* A method that is used at various places throughout the program to reformat a
 * date value when its textbox control loses focus. This is a convenience to the
 * user because it allows the user to enter only part of the date, or a letter
 * indicating, for example, today or yesterday, and the program will flesh out
 * the value to a complete date value.
 * The method returns the reformatted date value unless it does not recognize
 * the input as a valid pattern, in which case it simply returns the value that
 * is already in the textbox.
 *
 * Reformatting that is done by this method:
 * - If the user enters a t or T, the current date is returned.
 * - If the user enters a y or Y, yesterday's date is returned.
 * - If the user enters a t or T followed by a - or + and one to three digits,
 *   the method will calculate the number of days from the current date that
 *   the operation represents, and returns that date. For example, t-2 will
 *   return the date that was two days ago.
 * - If the user enters only the month and day, separated by a slash, the
 *   method will automatically attach the second slash and the current year.
 *   For example, if it's 2011 and the user enters 2/1, the method will
 *   return 2/1/2011.
 */

// The regular expression that is used to determine whether the user has entered
// an expression for a number of days before or after the current date (e.g., t-2).
    string regex1 = "^T(-|\\+)\\d{1,3}$";

    // Fields used in the logic that determines whether the user has entered a partial
    // date that has to be augmented with the current year.
    string[] dateString;
    string datePart;

    // Holds the date value if it's been reformatted from a partial date.
    string reformattedDate;

    // Trim any leading and trailing spaces.
    parText = parText.Trim();


    // Begin reformatting logic.
    // If a t or T was entered, return today's date.
    if (parText.ToUpper() == "T") {
        parText = DateTime.Today.ToShortDateString();
    }
    // If a y or Y was entered, return yesterday's date.
    else if (parText.ToUpper() == "Y") {
        parText = DateTime.Today.AddDays(-1).ToShortDateString();
    }
    // If an expression based on today's date (e.g., t-2) was entered, calculate and return
    // the appropriate date.
    else if (Regex.IsMatch(parText.ToUpper(), regex1)) {
        parText = DateTime.Today.AddDays(Int32.Parse(parText.Substring(1))).ToShortDateString();
    }
    else {
        // Otherwise, determine whether it's a partial date that needs to be fleshed out.
        // Split the entered date into multiple strings, using the slash as the delimiter.
        dateString = parText.Split('/');
        reformattedDate = string.Empty;
        // The entered value can possibly be a date only if there are at least two parts and no
        // more than three parts. Anything else is not a valid date, so will not be reformatted
        // by this method.
        if (dateString.Length > 1 && dateString.Length < 4) {
            // Loop through the date parts returned by the Split functions.
            for (int x = 0; x < dateString.Length; x++) {
                datePart = dateString[x].Trim();            // Trim any leading and trailing spaces.
                if (x < 2) {                                // If the month or day portion,
                    if (datePart.Length < 1) {              // if the length is less than one,
                        break;                              // it's not a valid date.
                    }
                    else {
                        datePart += "/";                    // Otherwise, append a slash.
                    }
                }
                reformattedDate += datePart;      // Append the date part to the reformatted date variable.
            }
            // If the reformatted date value ends with a slash and has more than one slash, it looks like
            // we've got a valid month/day value, and we can append the current year.
            // (The program determines the number of slashes by subtracting the length of the value without
            // slashes from the length of the value with slashes.)
            if (reformattedDate.EndsWith("/") && (reformattedDate.Length - reformattedDate.Replace("/",
"").Length > 1)) {
                reformattedDate += DateTime.Today.Year.ToString();
            }
            parText = reformattedDate;
        }
    }
    return parText;     // Return the value, whether reformatted or not.
}
```

Phone Number Shortcut

This method takes a string as a parameter and, if it recognizes it as the basis for a valid phone number, reformats it into a complete phone number value and returns it. Phone numbers in this application are stored in nnn-nnn-nnnn format. Any other characters (for example, parentheses around the area code) are discarded.

Thus, the user need key in only the digits, but including other characters will still result in a proper reformatting, provided a recognizable number of digits has been entered.

```
public string CommonPhoneTextLeaveHandler(string parText) {
    /* A method that is used at various places throughout the program to reformat a
     * phone number value when its textbox control loses focus. This is a convenience
     * to the user so the user doesn't have to enter the phone number in the exact
     * format that the program expects.
     * The method will reformat a phone number value that has eleven digits
     * (e.g., 1-555-666-7777), ten digits (555-666-7777), or seven digits (666-7777).
     * Any non-digit character is ignored (for example, if the user enters parentheses
     * around the area code, they will be ignored). The method returns a value with
     * hyphens separating the major portions of the phone number. (So if parentheses
     * have been entered, they will be discarded.)
     * If the phone number value supplied does not contain eleven, ten, or seven digits,
     * the phone number will not be reformatted, and the original value is returned
     * instead.
     */

    string phoneDigits;             // Holds only the digits from the phone number.
    int phoneNumberLength;
    StringBuilder phoneString;

    // Strip out everything but the digits.
    phoneDigits = Regex.Replace(parText, "[^0-9]", "");

    // Save the length of the number so we can use that value several times instead of
    // having to retrieve the .Length property every time we need the length.
    phoneNumberLength = phoneDigits.ToString().Length;

    // Prepare the StringBuilder object.
    phoneString = new StringBuilder();
    phoneString.Remove(0, phoneString.Length);

    // We will proceed with reformatting the number only if one of the following is true:
    // - The phone number is eleven digits long and begins with a 1.
    // - The phone number is ten digits long.
    // - The phone number is seven digits long.
    if ((phoneNumberLength == 11  &&  parText.Substring(0,1) == "1")  ||  phoneNumberLength == 10  ||
phoneNumberLength == 7) {
        // If the length of the number is eleven, append its first character to the reformatted
        // number being created, remove that first character from the digits-only phone number
        // variable, and append a hyphen to the phone number being created.
        if (phoneNumberLength == 11) {
            phoneString.Append(phoneDigits.Substring(0, 1));
            phoneDigits = phoneDigits.Remove(0,1);
            phoneString.Append("-");
        }
        // If the length of the remainder of the phone number is more than seven (i.e., is ten),
        // extract the first three digits, append them to the phone number being created, then
        // append a hyphen to the phone number being created.
        if (phoneNumberLength > 7) {
            phoneString.Append(phoneDigits.Substring(0,3));
            phoneDigits = phoneDigits.Remove(0, 3);
            phoneString.Append("-");
        }
        // At this point, there should just be seven digits left. Append the first three digits
        // to the number being created, followed by a hyphen, then the last four digits.
        phoneString.Append(phoneDigits.Substring(0,3));
        phoneString.Append("-");
        phoneString.Append(phoneDigits.Substring(3));
    }
    // If a reformatted phone number was created, assign it to the variable that will be
    // returned by the method.
    if (phoneString.Length > 0) {
        parText = phoneString.ToString();
    }
    return parText;     // Return the phone number, either in its original format or reformatted.
}
```

Validation with Regular Expression

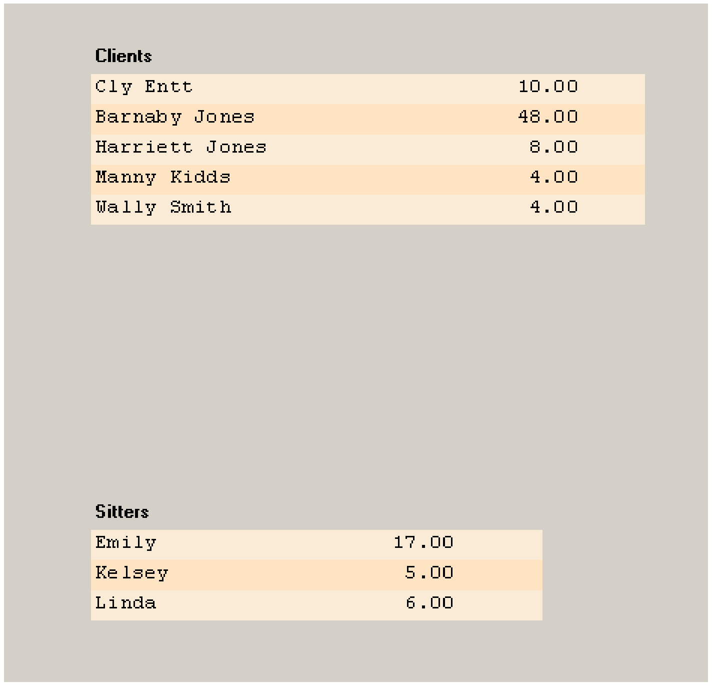
Validation of some types of data entry field that are used in several places throughout the application was coded in common methods that could be used for any such field. One such data type is the currency type, used for recording various customer charges. The method in this section performs such validation. It takes a string value as a parameter, and a second parameter that specifies whether or not for this instance a negative value should be permitted. The method returns a Boolean value indicating validation success or failure, as well as the charge amount as a decimal value, the latter via an out parameter. (If the source value is not a valid currency value, the returned decimal value will be zero.)

A regular expression is used in this validation. The regular expression is modified slightly within the method if the option for permitting negative values is used.

```
public Boolean ValidateChargeAmount(string parCharge, Boolean parAllowNegatives, out decimal chargeAmount)
{

    // Overloaded method for validating a currency amount. This version supports validating negative
    // values; whether a negative value is allowed depends on the value of the allow negatives
    // parameter.

    chargeAmount = 0M;

    // The value supplied to the method is considered good unless one of the validation expressons
    // proves otherwise.
    Boolean goodValue = true;

    // This regular expression string matches on a value that has;
    // - Zero to three digits to the left of a period.
    // - Zero or one period.
    // - Zero or two digits following a period.
    // - The ^ and $ that encompass the expression mean that the value must contain no other characters
    //   (i.e., it must begin and end with the expression).
    string regex = "^\\d{0,3}\\.?\\d{0,2}$";

    // If the allow negatives parameter is true, an optional negative sign is appended to the beginning
    // of the regular expression string.
    if (parAllowNegatives) {
        regex = regex.Replace("^", "^-?");
    }

    // Because the regular expression in theory would match on an empty string (because each component
    // of the regular expression is set to allow at minimum zero occurrences), the method must check
    // to see if the supplied value is of zero length. If it is, the value is deemed invalid.
    if (parCharge.Length < 1) {
        goodValue = false;
    }
    else {
        // Otherwise, try to parse the value as a decimal object.
        if (Decimal.TryParse(parCharge, out chargeAmount)) {
            // If successful, match it against the regular expression.
            if (Regex.IsMatch(parCharge, regex)) {
            }
            else {
                goodValue = false;          // The value did not match the regular expression.
            }
        }
        else {
            goodValue = false;              // The value did not parse as a decimall object.
        }
    }
    return goodValue;
}
```

ListBox with varying background color

The background row colors of the client list and the sitter list on the Taxes page alternate to make viewing the information easier. The portion of that page with those lists is shown at right. Note that these are ListBox controls that will display a vertical scroll bar if there are more rows than the control can display at once.


Clients

Cly Entt	10.00
Barnaby Jones	48.00
Harriett Jones	8.00
Manny Kidds	4.00
Wally Smith	4.00

Sitters

Emily	17.00
Kelsey	5.00
Linda	6.00

The alternating row color is not a native feature of the ListBox control, but it can be done by taking control of some of the ListBox drawing operations. To illustrate how this was done, the code that prepares the client ListBox for this capability is presented, followed by the two even handlers that are called when each row is added to the ListBox. (The same approach is used also for the sitter ListBox, and the event handlers are common, called by both the client ListBox code and the sitter ListBox code.)The following statements appear in method FillTaxesClientListBox(), which loads the client ListBox items. The first statement sets the ListBox's DrawMode, which allows the deviation from standard ListBox properties to occur. The second statement specifies the event handler to use when each item is added to the ListBox. And the third line sets the ListBox row height, in this case larger than the default.

```
// In order to use a different BackColor for each row in the ListBox, the DrawMode of the ListBox must be
// set to OwnerDrawVariable. (OwnerDrawVariable is used instead of OwnerDrawFixed so that the height of
// the rows can be altered from the default height value.)
lstTaxesClients.DrawMode = DrawMode.OwnerDrawVariable;

// A DrawItemEventHandler must be used in order to effect the variable BackColor-by-row feature.
lstTaxesClients.DrawItem += new DrawItemEventHandler(ListBox_DrawItem);
```

```
// This handler is needed so that the row height can be set to the desired value.
lstTaxesClients.MeasureItem += new MeasureItemEventHandler(ListBox_MeasureItem);
```

The method below is the event handler that is called as each row is added to the ListBox.

```
private void ListBox_DrawItem(object sender, DrawItemEventArgs e) {
    // This is a generic handler that allows the default characteristics of the rows of a ListBox in
    // OwnerDrawVariable DrawMode to be overridden. In this case, it is used to specify alternating
    // colors for the row background, and to specify a monospaced font.


    // The DrawItemEventArgs are modified:
    // - The DrawItemState is set to the Default state so that the background color doesn't change if the
    //   item is selected (which happens even if the ListBox SelectionMode is set to None).
    // - The background color of the line item is alternated between two colors to make the reading of each
    //  line easier.

    e = new DrawItemEventArgs(e.Graphics, e.Font, e.Bounds, e.Index, DrawItemState.Default, e.ForeColor,
(e.Index % 2 == 0) ? Color.AntiqueWhite : Color.Bisque);


    // Draw the background of the ListBox control for each item.
    e.DrawBackground();

    // Draw the current item text using a mono-spaced font.
    e.Graphics.DrawString(((ListBox)sender).Items[e.Index].ToString(), new
Font(FontFamily.GenericMonospace, 10), Brushes.Black, e.Bounds, StringFormat.GenericDefault);

    // If the ListBox has focus, draw a focus rectangle around the selected item.
    e.DrawFocusRectangle();
}
```

This method is the one that controls the ListBox line height.

```
private void ListBox_MeasureItem(object sender, MeasureItemEventArgs e) {
    // This is a generic handler that allows a ListBox in OwnerDrawVariable DrawMode to have a row height
    // that is other than the default height.
    e.ItemHeight = 20;
}
```

Total hour calculation

The total hours figure displayed on the Taxes page is used for calculating the portion of household expenses such as utilities can be considered a business expense. Calculation involves finding the longest appointment duration for each appointment type on each day, and adding that duration to the total number of minutes for the time period being reported. For example, if on March 4 two children were cared for before school—one from 7:00-8:00, the other from 7:15-8:00—then the number of minutes for the before school appointment type on March 4 is 60, and that number is added to the running total. This sort of calculation is done for each appointment type on every day during the year(s) being reported.

```
public void CalculateTotalHours() {

    /* This method calculates the number of hours that the child care business was used during the
     * selected reporting period. This is done by retrieving the appointment records (only those
     * that were approved, because they're the only ones that reflect actual appointments) for the
     * reporting period. For each appointment type (before school or after school) on a given date,
     * the longest appointment period (end time - start time) is used. For example, if there were
     * three children before school on September 10--two that were there for 45 minutes and one that
     * was there for 35 minutes--the appointment period would be the longest one, namely, 45 minutes.
     * Those 45 minutes are added to the total. This computation is repeated for each appointment
     * type on each day that there was an appointment.
     */

    StringBuilder sqlString;
    OleDbCommand dbCommand;
    OleDbDataAdapter da;
    DataSet ds;
    DataRowCollection dataRows;
    DateTime apptDate = new DateTime();
    DateTime startTime = new DateTime();
    DateTime endTime = new DateTime();
    TimeSpan timeDiff;
    decimal totalMinutes;
    Boolean dataIsGood = new bool();
    decimal totalHours;

    Boolean firstRow;
    DateTime lastApptDate = new DateTime();
```

```
int lastApptType = 0;
decimal lastMinutes = 0M;
decimal apptMinutes;
int dateCount = 0;
int apptType = 0;


// Build the SQL query for the selected time period.
// Note that only approved appointments are included.
// Also note the sequence in which the returned rows are ordered. This is essential in order for the
// logic that sums the hours to be accurate.
sqlString = new StringBuilder();
sqlString.Remove(0, sqlString.Length);
sqlString.Append("SELECT");
sqlString.Append(" tblAppointment.apptDate");
sqlString.Append(", tblAppointment.apptType");
sqlString.Append(", tblAppointment.actualStartTime");
sqlString.Append(", tblAppointment.actualEndTime");
sqlString.Append(" FROM tblAppointment");
sqlString.Append(" WHERE (");
sqlString.Append("(tblAppointment.apptDate BETWEEN #" + begDate + "# AND #" + endDate + "#)");
sqlString.Append(" AND (tblAppointment.approved <> 0)");
sqlString.Append(")");
sqlString.Append(" ORDER BY tblAppointment.apptDate, tblAppointment.apptType");

// Execute the query.
dbCommand = new OleDbCommand(sqlString.ToString(), Program.DBconn);
da = new OleDbDataAdapter(dbCommand);
ds = new DataSet();
Program.DBconn.Open();
da.Fill(ds, "dsApptData");
Program.DBconn.Close();
dataRows = ds.Tables["dsApptData"].Rows;

// This variable is needed because there will have been no prior appointment date and appointment type
// to which those in the first row can be compared. This variable be set to false after the first row
// has been processed.
firstRow = true;

// The variable that holds the sum of all the minutes. We must start at zero.
totalMinutes = 0M;
```

```
// Loop through the dataset rows. The basic concept is that if the duration (number of minutes) from
// the appointment currently being examined is greater than that from the prior appointment, the
// minutes from the current one are saved, replacing the previous value.
// This continues until there is a break in appointment date or appointment type. (There can be
// multiple appointment types within a given appointment date, so there may an appointment type break
// between appointment date breaks.) When such a break occurs, we know we've obtained the largest
// duration for the prior date/type combination, and we add that last saved number of minutes to the
// total number of minutes. Any time there has been a break on appointment type, we save the
// appointment type from the new row for comparison to the value in the next row, and we save the
// number of minutes from the new row as the new "greatest duration encountered" value. (The minutes
// from the next row will be compared to this "greatest duration" value if the date and type are the
// same; if the minutes in the next row are greater than the prior on, the minutes from the new row
// will become the "greatest duration encountered" value; otherwise, it will remain unchanged.)
// Similarly, when there is a break on appointment date, the appointment date and appointment type are
// both saved for comparison to the next row, as is its minutes value.

foreach (DataRow dr in dataRows) {
    // The program ensures that the values in the database row are valid. If any column value is found
    // to be invalid, the row is rejected. The variable below is used to specify whether an invalid
    // value has been encountered, and is set to true before validation is begun. Each succeeding
    // validation is done only so long as the program has not encountered a bad value. It's unnecessary
    // to do additionaly validation after a bad value has been found because the row will be rejected
    // if any of its values is bad.
    dataIsGood = true;
    if (dataIsGood) {
        if (!DateTime.TryParse(dr["apptDate"].ToString(), out apptDate)) {
            dataIsGood = false;
        }
    }
    if (dataIsGood) {
        if (!int.TryParse(dr["apptType"].ToString(), out apptType)) {
            dataIsGood = false;
        }
    }
    if (dataIsGood) {
        if (!DateTime.TryParse(dr["actualStartTime"].ToString(), out startTime)) {
            dataIsGood = false;
        }
    }
```

```
if (dataIsGood) {
    if (!DateTime.TryParse(dr["actualEndTime"].ToString(), out endTime)) {
        dataIsGood = false;
    }
}

// End of validation.

if (dataIsGood) {
    timeDiff = endTime.Subtract(startTime);                     // Get the difference between the
                                                                // row's end time and start time.
    apptMinutes = Convert.ToDecimal(timeDiff.TotalMinutes);     // Convert that difference to
                                                                // minutes.
    if (apptDate != lastApptDate) {                 // Appointment date break.
        dateCount++;                                // Count the number of appointment dates so that
                                                    // number can be displayed.
        if (firstRow) {                             // No comparison to prior row if this is the first
                                                    // row being processed.
            firstRow = false;
            lastApptDate = apptDate;
            lastApptType = apptType;
            lastMinutes = apptMinutes;
        }
        else {
            totalMinutes += lastMinutes;            // Add the last saved minutes to the total minutes.
            lastApptDate = apptDate;                // Save the appointment date for comparison.
            lastApptType = apptType;                // Save the appointment type for comparison.
            lastMinutes = apptMinutes;              // Save the minutes for comparison and possible
                                                    // addition to total minutes.
        }
    }
    else {
        if (apptType != lastApptType) {             // Appointment type break;
            totalMinutes += lastMinutes;            // Add the last saved minutes to the total minutes.
            lastApptType = apptType;                // Save the appointment type for comparison.
            lastMinutes = apptMinutes;              // Save the minutes for comparison and possible
                                                    // addition to total minutes.
        }
        else {                                      // Otherwise, appointment date and type have not
                                                    // changed.
            if (apptMinutes > lastMinutes) {        // If the number of minutes from the current row is
```

```
                                                    // greater than the prior row,
                    lastMinutes = apptMinutes;      // save the number from the current row.
                }
            }
        }
    }
}

// The last row will not have been added to the total, so do so now.
// But only do it if the last row does not contain invalid data.

if (dataIsGood) {
    totalMinutes += lastMinutes;
}

// Display the computed information.
lblTaxesDaysUsed.Text = dateCount.ToString();
totalHours = totalMinutes / 60;
lblTaxesAvgHours.Text = dateCount > 0 ? (totalHours / (Decimal)dateCount).ToString("0.00") : "0";
lblTaxesTotalHours.Text = totalHours.ToString("#,#.00");
}
```

Index retrieval for newly-inserted database records

In a couple circumstances it is necessary to have access to the auto-generated index of a newly-inserted database record. This was a function that I couldn't recall doing ever, so I searched the Web for to see if it was possible with an Access database. The solution was to issue a SELECT @@IDENTITY statement using the ExecuteScalar() database method. The example below is from the code that inserts a new row into the tblFamily database.

```
public int AddFamily()
{
    int familyNumber;

    StringBuilder sqlString = new StringBuilder();
    sqlString.Remove(0, sqlString.Length);
    sqlString.Append("INSERT INTO tblFamily (");
    sqlString.Append("familyName");
    sqlString.Append(",firstClientNumber");
    sqlString.Append(",secondClientNumber");
    sqlString.Append(",percentageBased");
    sqlString.Append(",firstClientBaseRateBefore");
    sqlString.Append(",firstClientBaseRateAfter");
    sqlString.Append(",firstClientPercentage");
    sqlString.Append(",secondClientBaseRateBefore");
    sqlString.Append(",secondClientBaseRateAfter");
    sqlString.Append(",secondClientPercentage");
    sqlString.Append(",notes");
    sqlString.Append(") VALUES (");
    sqlString.Append("'" + familyName + "'");
    sqlString.Append("," + firstClientNumber);
    sqlString.Append("," + secondClientNumber);
    sqlString.Append("," + percentageBased);
    sqlString.Append("," + firstClientBaseRateBefore);
    sqlString.Append("," + firstClientBaseRateAfter);
    sqlString.Append("," + firstClientPercentage);
    sqlString.Append("," + secondClientBaseRateBefore);
    sqlString.Append("," + secondClientBaseRateAfter);
    sqlString.Append("," + secondClientPercentage);
    sqlString.Append(",'" + notes + "'");
    sqlString.Append(")");

    OleDbCommand dbCommand = new OleDbCommand(sqlString.ToString(), Program.DBconn);
    Program.DBconn.Open();
    dbCommand.ExecuteNonQuery();
    dbCommand.CommandText = "SELECT @@IDENTITY";        // Get the index of the record that was just added
                                                        // to the database.
    familyNumber = (int)dbCommand.ExecuteScalar();
    Program.DBconn.Close();

    return familyNumber;
}
```

Building the Calendar

The calendar is undoubtedly one of the most complex structures in the program. There are several methods responsible for building its various parts: The cells and their dates, the appointments, and the sitter lists. I wanted to be able to treat the cells and the data therein as iterative items so they could be processed via loops. Through some research on the Web I discovered a way to define an array of form controls. Using this approach, an array of Panel objects was created, each being one of the calendar cells. (The rows on the Ledger page were constructed using the same approach.)

This section of the document presents the constructor and property that define the array of cell panels, followed by the method that creates those cells. Finally, one of the methods that loads the calendar cells with appointments is presented. (The loading of the calendar dates and the loading of the sitter lists are done in other methods not included here.)

```
/*** CONSTRUCTORS ***/

public CalendarArray(TabPage host)
{
    HostTab = host;
}



/*** PROPERTIES ***/

public Panel this[int Index]
{
    get
    {
        return (Panel)this.List[Index];
    }
}



/*** BUSINESS LOGIC METHODS ***/
```

```
public void AddNewCell(int x)
{
    Panel calPanel = new Panel();                           // Create a new instance of the Panel class for the
                                                            // current calendar cell.
    this.List.Add(calPanel);                                // Add the Panel to the collection's list.
    HostTab.Controls.Add(calPanel);                         // Add the Panel to the controls collection of the form
                                                            // referenced by the HostTab field.


    // Set the intial properties for the Panel object.
    calPanel.BackColor = Color.White;
    calPanel.BorderStyle = BorderStyle.FixedSingle;
    calPanel.Height = cellHeight;

    // The calendar shows three weeks, seven days to a row.
    // Determine the row on which the cell being processed belongs.
    if (x < 7)
    {
        currentRow = 0;
    }
    else if (x < 14)
    {
        currentRow = 1;
    }
    else
    {
        currentRow = 2;
    }

    // Determine the vertical location of the top of the cell being created.
    calPanel.Top = calTop + (currentRow * cellHeight) - (currentRow * 1);

    // The width of the calendar cell is now determined. Because appointments cannot be made
    // on weekend days, the cells for the weekend days are narrower than they are for weekdays.
    if (FormMain.weekendDates.Contains(x)) {
        calPanel.Width = 60;
    }
    else {
        calPanel.Width = 180;
    }
    // The left edge of the cell is now determined. For the first cell on each row, it will
    // be the left edge of the calendar. Otherwise, it will be the cumulative widths of the
    // cells on that row so far.
    switch (x)
    {
        case 0:
        case 7:
        case 14:
            calPanel.Left = calLeft;
            break;
        default:
            calPanel.Left = nextLeft;
            break;
    }

    // Compute the left edge of the next panel.
    // One is subtracted at the end to make the borders overlap. Without that, the
    // border line appears too thick.
    nextLeft = calPanel.Left + calPanel.Width - 1;
}




public void LoadAppointments(Panel parPanel) {

    /* This method loads the appointments for a given calendar cell, specified as a parameter
     * to the method. It also calls the method that adds the before school and after school
     * sitter ComboBoxes to the calendar cell.
     */

    ApptTypes apptType;
    int apptTypeNum;
    string calDate;
    StringBuilder sqlString = new StringBuilder();
    DateTime actualStartTime;
    DateTime actualEndTime;

    // If there was an existing ListView for this calendar cell, remove it; it will be
    // created again for the date that this cell now represents. Note that the ListView
    // must be cleared before it can be removed.
    if (parPanel.Controls.Contains(parPanel.Controls["lvCalendarApptList"])) {
        // can't remove listview control if it has an item selected!!!
        ((ListView)parPanel.Controls["lvCalendarApptList"]).Clear();
        parPanel.Controls.Remove(parPanel.Controls["lvCalendarApptList"]);
    }

    // Create a new ListView object for this calendar cell.
    ListView lvCalendarApptList = new ListView();
    lvCalendarApptList.Top = calListViewTop;
    lvCalendarApptList.Width = parPanel.Width - 4;
    lvCalendarApptList.Height = parPanel.Height - calListViewTop - 28;
    // Same background color as the parent cell.
    lvCalendarApptList.BackColor = parPanel.BackColor;
    lvCalendarApptList.BorderStyle = BorderStyle.None;
    lvCalendarApptList.Name = "lvCalendarApptList";
    // Can select only one item at a time.
    lvCalendarApptList.MultiSelect = false;
    // Can click anywhere on the row to select it.
    lvCalendarApptList.FullRowSelect = true;
    lvCalendarApptList.View = View.Details;
    lvCalendarApptList.Font = new Font("Microsoft Sans Serif", 8F, FontStyle.Regular);
    lvCalendarApptList.HeaderStyle = ColumnHeaderStyle.None;
    // Need only one click to select an item.
    lvCalendarApptList.Activation = ItemActivation.OneClick;
    lvCalendarApptList.TabStop = false;
    lvCalendarApptList.ItemActivate += new EventHandler(lvCalendarApptList_ItemActivate);
    // Must add columns to a ListView in Details view, or the data won't show up.
    lvCalendarApptList.Columns.Add("", 0);
    lvCalendarApptList.Columns.Add("", 86);
    lvCalendarApptList.Columns.Add("", 90);

    parPanel.Controls.Add(lvCalendarApptList);      // Add the ListView to the calendar cell.

    // Get the date that is in the Tag property of the calendar cell being set up.
    calDate = ((DateTime)parPanel.Tag).ToString("yyyy/MM/dd");

    // Construct an SQL query for retrieving the appointments for this date.
    sqlString.Remove(0, sqlString.Length);
    sqlString.Append("SELECT");
```

```
    sqlString.Append(" tblAppointment.apptNumber");
    sqlString.Append(", tblAppointment.childNumber");
    sqlString.Append(", tblAppointment.actualStartTime");
    sqlString.Append(", tblAppointment.actualEndTime");
    sqlString.Append(", tblAppointment.apptType");
    sqlString.Append(", tblAppointment.approved");
    sqlString.Append(", tblChild.firstName + ' ' + LEFT(tblChild.lastName,1) AS childName");
    sqlString.Append(" FROM tblAppointment, tblChild");
    sqlString.Append(" WHERE (tblAppointment.apptDate BETWEEN #" + calDate + "# AND #" + calDate + "#)");
    sqlString.Append(" AND tblAppointment.childNumber = tblChild.childNumber");
    sqlString.Append(" ORDER BY tblAppointment.actualStartTime, tblAppointment.apptType, tblChild.lastName,
tblChild.firstName");

    // Execute the query.
    DataSet ds = new DataSet();
    OleDbCommand dbCommand = new OleDbCommand(sqlString.ToString(), Program.DBconn);
    OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
    Program.DBconn.Open();
    da.Fill(ds, "tblAppointment");
    Program.DBconn.Close();

    // For each row returned, add an item to the appointment ListView.
    DataRowCollection dataRows = ds.Tables["tblAppointment"].Rows;
    foreach (DataRow dr in dataRows) {
        // The appointment number is the main item in the ListViewItem.
        ListViewItem lvItem = new ListViewItem(dr["apptNumber"].ToString());

        // Attach the approved flag to the ListViewItem.
        // Highlight a ListView row that has been approved so it is apparent to the user.
        lvItem.Tag = dr["approved"];
        if ((bool)dr["approved"]) {
            lvItem.BackColor = Color.LightGoldenrodYellow;
        }

        // The first subitem added to the ListViewItem is the child name.
        lvItem.SubItems.Add((string)dr["childName"]);

        // Ensure that the start date and end date can be converted to DateTime objects, then format
        // the times for display.
        DateTime.TryParse(dr["actualStartTime"].ToString(), out actualStartTime);
        DateTime.TryParse(dr["actualEndTime"].ToString(), out actualEndTime);
```

```
            lvItem.SubItems.Add(FormatApptTimespan(actualStartTime, actualEndTime));

            // Parse the appointment type value to an integer, then convert it to an ApptType enumeration.
            // The color of the the ListView row varies by appointment type.
            int.TryParse(dr["apptType"].ToString(), out apptTypeNum);
            apptType = NumToEnum<ApptTypes>(Int32.Parse(dr["apptType"].ToString()));
            lvItem.ForeColor = apptType == ApptTypes.Before ? beforeSchoolColor : afterSchoolColor;

            // Add the item to the ListView.
            ((ListView)parPanel.Controls["lvCalendarApptList"]).Items.AddRange(new ListViewItem[] { lvItem });
        }

        // Load the sitter ComboBoxes to this calendar panel.
        LoadSitterComboBoxes(parPanel);
    }
```

Load the Ledger

This lengthy method illustrates the logic needed for loading the rows on the Ledger page. Because a series of Panel objects—and not a ListBox or ListView control—was used for the rows, it was necessary to accurately position all the controls on each row. The most complex part of the row build is the setup of the client controls; they are tailored to whether there is one client only, two clients who pay by percentage, or two clients who do not pay by percentage.

It is in this method that any deviation from the standard appointment charge is calculated for those occasions when a child has stayed for a duration that varies from the normal appointment stay. The method also stores the ledger row information in a List object, which is accessed during the data validation process. Those list items are filled as the rows are built; when the method that records all the information for approved appointments is run, the information is then easily accessed from the List instead of from the form controls, which in some cases hold a name instead of the table key value needed when writing the new database records.

```
public void FillAppointmentLedger() {
    /* This method loads both the ledger rows displayed on the screen and the LedgerRowInfo items
     * that correspond to those rows.
     */

    DateTime apptDate;
    DateTime standardStartTime;
    DateTime standardEndTime;
    DateTime actualStartTime;
    DateTime actualEndTime;
    TimeSpan startDiff;
    TimeSpan endDiff;
    int apptNumber;
    int apptTypeNumber;
    int familyNumber;
    int minuteDiff;
    int sitterNumber;
    decimal incrementCount;
    decimal extraCharge;
    decimal chargeAmountFirstClient;
    decimal totalChargeFirstClient;
    decimal chargeAmountSecondClient;
    decimal totalChargeSecondClient;
    ApptTypes apptType;
    string apptDateAndType;
    ClientType clientType;
    Family currentFamily = new Family();
    Client firstClient;
    Schedule sched;
    StringBuilder sqlString;

    // Hide the label that shows the error message when a row has invalid data.
    lblLedgerError.Visible = false;


    // If the array that holds the ledger row panels exists (it will not exist the first time
    // the Ledger tab page is displayed), then the ledger Panel objects are removed so we
    // can create new ones from scratch.
    if (ledgerArray != null) {
        TabPage tabPage = ((TabPage)ledgerArray[0].Parent);
        for (int x = 0; x < ledgerRowCount; x++) {
            tabPage.Controls.Remove(ledgerArray[x]);
        }
    }

    // Create a new ledger array.
    ledgerArray = new LedgerArray(tabPageLedger);

    // Add the Panel objects, one per ledger array item.
    for (int x = 0; x < ledgerRowCount; x++) {
        ledgerArray.AddNewLedgerRow(x);
    }

    // The list of ledger row data is cleared out to get ready for a new set.
    ledgerRows.RemoveRange(0, ledgerRows.Count);

    // Construct an SQL query to get the first n appointments, where n is the number of ledger rows
    // that fit on the tab page.
    // Note that the date comparison in the WHERE clause options is attached only if the approve
    // future appointments configuration item is not true, which means the approval of future
    // appointments is not allowed.
    sqlString = new StringBuilder();
    sqlString.Remove(0, sqlString.Length);
```

```
    sqlString.Append("SELECT TOP " + ledgerRowCount + " *");
    sqlString.Append(", tblChild.firstName + ' ' + LEFT(tblChild.lastName,1) AS childName");
    sqlString.Append(" FROM tblAppointment, tblChild");
    sqlString.Append(" WHERE tblAppointment.childNumber = tblChild.childNumber");
    sqlString.Append(" AND tblAppointment.approved = 0");
    if (!cfgApproveFutureAppt) {
        sqlString.Append(" AND tblAppointment.actualEndTime <= #" + DateTime.Now + "#");
    }
    sqlString.Append(" ORDER BY tblAppointment.apptDate, tblAppointment.apptType, tblChild.lastName,
tblChild.firstName");

    // Execute the query.
    DataSet ds = new DataSet();
    OleDbCommand dbCommand = new OleDbCommand(sqlString.ToString(), Program.DBconn);
    OleDbDataAdapter da = new OleDbDataAdapter(dbCommand);
    Program.DBconn.Open();
    da.Fill(ds, "dsAppointment");
    Program.DBconn.Close();

    int row = 0;    // Used for specifying the ledger array item being created.

    DataRowCollection dataRows = ds.Tables["dsAppointment"].Rows;

    // The query results are now processed.
    foreach (DataRow dr in dataRows) {

        // Store the database row's values in variables of the correct type, using TryParse
        // calls to prevent any possible exceptions.
        DateTime.TryParse(((DateTime)dr["apptDate"]).ToShortDateString(), out apptDate);
        int.TryParse(dr["apptType"].ToString(), out apptTypeNumber);
        apptType = NumToEnum<ApptTypes>(apptTypeNumber);
        DateTime.TryParse(dr["standardStartTime"].ToString(), out standardStartTime);
        DateTime.TryParse(dr["standardEndTime"].ToString(), out standardEndTime);
        DateTime.TryParse(dr["actualStartTime"].ToString(), out actualStartTime);
        DateTime.TryParse(dr["actualEndTime"].ToString(), out actualEndTime);
        int.TryParse(dr["familyNumber"].ToString(), out familyNumber);

        // Get the difference between the standard start time and the actual start time,
        // as well as the standard end time and the actual end time.
        startDiff = standardStartTime.Subtract(actualStartTime);
        endDiff = actualEndTime.Subtract(standardEndTime);
```

```
// Calculate the number of additional minutes for which the client should be charged.
// Note that if the actual time of the appointment was less than the standard time,
// this will be a negative value.
minuteDiff = Convert.ToInt32(startDiff.TotalMinutes + endDiff.TotalMinutes);

// The number of increments represented by the additional minutes is calculated,
// and rounded off.
incrementCount = Math.Round((Decimal)minuteDiff / (Decimal)cfgMinuteIncrement);

// The extra charge amount is computed.
extraCharge = incrementCount * cfgIncrementCharge;

// Create a new LedgerRowInfo item for this ledger row. After all necessary data has
// been added to this LedgerRowInfo item, the item will be added to the LedgerRowInfo
// List.
LedgerRowInfo ledgerRow = new LedgerRowInfo();

// Initialize the LedgerRowInfo item's fields.
ledgerRow.RowIsChecked = false;
int.TryParse(dr["apptNumber"].ToString(), out apptNumber);
ledgerRow.ApptNumber = apptNumber;
ledgerRow.ApptDate = apptDate;
ledgerRow.ClientType = ClientType.None;
ledgerRow.SelectedClient = 0;
ledgerRow.FirstClientCharge = 0;
ledgerRow.SecondClientCharge = 0;

// The program now creates the Windows form controls for the current ledger row.

// ERROR LABEL.
// This is an asterisk that will be shown if the row has an invalid value. Its
// visibility is initially set to false.
Label lblError = new Label();
lblError.Name = "lblError";
lblError.Width = 10;
lblError.Top = 10;
lblError.Font = new Font("Microsoft Sans Serif", 10F, FontStyle.Bold);
lblError.ForeColor = Color.Red;
lblError.Text = "*";
lblError.Visible = false;
```

```
ledgerArray[row].Controls.Add(lblError);

// APPOINTMENT DATE LABEL.
// This label holds the row's appointment date.
Label lblLedgerRowDate = new Label();
lblLedgerRowDate.Name = "lblLedgerRowDate";
lblLedgerRowDate.Width = 65;
lblLedgerRowDate.Text = apptDate.ToShortDateString();
lblLedgerRowDate.Top = 11;
lblLedgerRowDate.Left = 11;
ledgerArray[row].Controls.Add(lblLedgerRowDate);

// APPOINTMENT TIME LABEL.
// Shows the actual start and end time of the appointment.
Label lblLedgerRowTimespan = new Label();
lblLedgerRowTimespan.Name = "lblLedgerRowTimespan";
lblLedgerRowTimespan.Width = 70;
lblLedgerRowTimespan.Text = FormatApptTimespan(actualStartTime, actualEndTime);
lblLedgerRowTimespan.Top = 11;
lblLedgerRowTimespan.Left = 80;
ledgerArray[row].Controls.Add(lblLedgerRowTimespan);

// CHILD NAME LABEL.
Label lblLedgerRowName = new Label();
lblLedgerRowName.Name = "lblLedgerRowName";
lblLedgerRowName.Width = 100;
lblLedgerRowName.Text = dr["childName"].ToString();
lblLedgerRowName.Top = 11;
lblLedgerRowName.Left = 160;
ledgerArray[row].Controls.Add(lblLedgerRowName);

// To determine which and how the clients are displayed requires the retrieval
// of the Family information associated with the child to which the ledger row
// belongs.
// If the child does not belong to a family, the family number will be zero.
currentFamily = new Family();
currentFamily = GetFamilyData(familyNumber);

// Store in the LedgerRowInfo item the first and second client numbers.
ledgerRow.FirstClientNumber = currentFamily.FirstClientNumber;
ledgerRow.SecondClientNumber = currentFamily.SecondClientNumber;
```

```
// Determine the client type based on the characteristics from the Family
// instance, then store in the LedgerRowInfo item.
if (currentFamily.SecondClientNumber == 0) {
    clientType = ClientType.OneClient;
}
else {
    if (currentFamily.PercentageBased == true) {
        clientType = ClientType.PercentageClients;
    }
    else {
        clientType = ClientType.TwoClients;
    }
}
ledgerRow.ClientType = clientType;

// Get the base charge for the two clients, which can vary based on the appointment
// type (before school or after school). If the appointment type is neither before
// school nor after school (which should never happen), those amounts are set to zero.
if (apptType == ApptTypes.Before) {
    chargeAmountFirstClient = currentFamily.FirstClientBaseRateBefore;
    chargeAmountSecondClient = currentFamily.SecondClientBaseRateBefore;
}
else if (apptType == ApptTypes.After) {
    chargeAmountFirstClient = currentFamily.FirstClientBaseRateAfter;
    chargeAmountSecondClient = currentFamily.SecondClientBaseRateAfter;
}
else {
    chargeAmountFirstClient = chargeAmountSecondClient = 0;
}

// The extra charge amount calculated earlier in this method is added to the
// base charge to get the total charge for each client.
totalChargeFirstClient = chargeAmountFirstClient + extraCharge;
totalChargeSecondClient = chargeAmountSecondClient + extraCharge;

// Get the Client table info for the first client on the ledger row.
firstClient = new Client();
firstClient = GetClientData(currentFamily.FirstClientNumber);

// CLIENT GROUPBOX.
```

```
        // The clients and their amounts due are displayed in a GroupBox. This is done because in the
        // case of the TwoClients ClientType, the user will use radio buttons to determin which of the
        // two clients is responsible for payment. To make all rows look the same regardless of the
        // ClientType, all client information is placed in a GroupBox.
        GroupBox groupBox = new GroupBox();
        groupBox.Name = "grpLedgerRow";
        groupBox.Top = 3;
        groupBox.Left = 275;
        groupBox.Height = 33;
        groupBox.Width = 465;
        groupBox.Text = "";
        ledgerArray[row].Controls.Add(groupBox);

        // FIRST CLIENT RADIO BUTTON.
        // Radio buttons are used only when the user has to select one of the two clients.
        if (clientType == ClientType.TwoClients) {
            RadioButton radLedgerRowFirstClient = new RadioButton();
            radLedgerRowFirstClient.Name = "radLedgerRowFirstClient";
            radLedgerRowFirstClient.Text = "";
            radLedgerRowFirstClient.Width = 12;
            radLedgerRowFirstClient.Left = 4;
            radLedgerRowFirstClient.Top = 7;
            radLedgerRowFirstClient.CheckedChanged += new
System.EventHandler(radLedgerRowFirstClient_CheckedChanged);
            groupBox.Controls.Add(radLedgerRowFirstClient);
        }

        // FIRST CLIENT NAME LABEL.
        Label lblFirstClientName = new Label();
        lblFirstClientName.Name = "lblLedgerRowFirstClientName";
        lblFirstClientName.Width = 120;
        lblFirstClientName.Text = firstClient.FullName;
        lblFirstClientName.Left = 16;
        lblFirstClientName.Height = 12;
        groupBox.Controls.Add(lblFirstClientName);
        lblFirstClientName.Top = lblFirstClientName.Parent.Top + 8;

        // FIRST CLIENT CHARGE AMOUNT TEXTBOX.
        TextBox txtLedgerRowFirstClientAmount = new TextBox();
        txtLedgerRowFirstClientAmount.Name = "txtLedgerRowFirstClientAmount";
        txtLedgerRowFirstClientAmount.Left = 140;
        txtLedgerRowFirstClientAmount.Width = 40;
        txtLedgerRowFirstClientAmount.Top = 9;
        // If there is only one client for the family, or there are two clients but they do not pay
        // on a percentage basis, then the first client amount is placed in the textbox.
        // Otherwise, if there are two clients and they pay on a percentage basis, then the amount
        // to be charged is the first client amount times that percentage.
        if (clientType == ClientType.OneClient || clientType == ClientType.TwoClients) {
            txtLedgerRowFirstClientAmount.Text = totalChargeFirstClient.ToString("N");
        }
        else if (clientType == ClientType.PercentageClients) {
            txtLedgerRowFirstClientAmount.Text = (totalChargeFirstClient *
(currentFamily.FirstClientPercentage / 100)).ToString("N");
        }
        txtLedgerRowFirstClientAmount.Leave +=new EventHandler(txtLedgerRowFirstClientAmount_Leave);
        groupBox.Controls.Add(txtLedgerRowFirstClientAmount);

        // SECOND CLIENT RADIO BUTTON.
        // Radio buttons are used only when the user has to select one of the two clients.
        if (clientType == ClientType.TwoClients) {
            RadioButton radLedgerRowSecondClient = new RadioButton();
            radLedgerRowSecondClient.Name = "radLedgerRowSecondClient";
            radLedgerRowSecondClient.Text = "";
            radLedgerRowSecondClient.Width = 12;
            radLedgerRowSecondClient.Left = 280;
            radLedgerRowSecondClient.Top = 7;
            radLedgerRowSecondClient.CheckedChanged += new
System.EventHandler(radLedgerRowSecondClient_CheckedChanged);
            groupBox.Controls.Add(radLedgerRowSecondClient);
        }
        // SECOND CLIENT NAME LABEL AND CHARGE AMOUNT TEXTBOX.
        // A second client name and charge amount textbox are displayed if the family does not
        // have only one client.
        if (clientType == ClientType.TwoClients || clientType == ClientType.PercentageClients) {
            Client secondClient = new Client();
            secondClient = GetClientData(currentFamily.SecondClientNumber);
            Label lblSecondClientName = new Label();
            lblSecondClientName.Name = "lblLedgerRowSecondClientName";
            lblSecondClientName.Width = 120;
            lblSecondClientName.Text = secondClient.FullName;
            lblSecondClientName.Left = 292;
            lblSecondClientName.Height = 12;
```

```
                groupBox.Controls.Add(lblSecondClientName);
                lblSecondClientName.Top = lblSecondClientName.Parent.Top + 8;

                TextBox txtLedgerRowSecondClientAmount = new TextBox();
                txtLedgerRowSecondClientAmount.Name = "txtLedgerRowSecondClientAmount";
                txtLedgerRowSecondClientAmount.Left = 416;
                txtLedgerRowSecondClientAmount.Width = 40;
                txtLedgerRowSecondClientAmount.Top = 9;
                txtLedgerRowSecondClientAmount.Text = currentFamily.SecondClientBaseRateBefore.ToString("N");
                // The charge amount displayed is the total charge amount if the clients do not pay on a
                // percentage basis. Otherwise, the amount is the second client's percentage of the total
                // charge, which is stored in the total charge variable for the first client.
                if (clientType == ClientType.TwoClients) {
                    txtLedgerRowSecondClientAmount.Text = totalChargeSecondClient.ToString("N");
                }
                else if (clientType == ClientType.PercentageClients) {
                    txtLedgerRowSecondClientAmount.Text = (totalChargeFirstClient *
(currentFamily.SecondClientPercentage / 100)).ToString("N
                }
                txtLedgerRowSecondClientAmount.Leave += new EventHandler(txtLedgerRowSecondClientAmount_Leave);
                groupBox.Controls.Add(txtLedgerRowSecondClientAmount);
            }

            // To get the name of the sitter to display on the ledger row, the program will see if an entry
            // exists in the Dictionary that holds sitter numbers for a given appointment date/type
            // combination. If that date/type combination does not exist in the Dictionary -- because that
            // combination has not been encountered yet -- the sitter number is obtained from the database.
            // If obtained from the database, an entry is then added to the Dictionary so that future
            // retrievals for that date/time combination can be done from memory instead of from disk. This
            // approach is taken simply as a means to make the sitter number retrieval as efficient as possible
            // (i.e., with as few disk reads as possible).
            //
            // To look up the date/time combination in the Dictionary, the program concatenates the two values.
            apptDateAndType = apptDate.ToShortDateString() + ((int)apptType).ToString();

            // The program checks to see if the date/time combination exists in the Dictionary. If so, the
            // sitter number will have been placed in the sitterNumber variable. If not, the program retrieves
            // it from the database, then adds a corresponding entry in the Dictionary.
            if (!schedules.TryGetValue(apptDateAndType, out sitterNumber)) {
                sched = new Schedule();
                sched.GetSchedule(apptDate, apptType);
```

```
        sitterNumber = sched.SitterNumber;
        if (sitterNumber > 0) {
            schedules.Add(apptDateAndType, sitterNumber);
        }
    }

    // Add the sitter number to the LedgerRowInfo item for the current ledger row.
    ledgerRow.SitterNumber = sitterNumber;

    // SITTER NAME LABEL.
    // The sitter name is obtained from the list of sitters that was loaded into memory when the Ledger
    // tab page was first displayed.
    Label lblSitterName = new Label();
    lblSitterName.Name = "lblSitterName";
    lblSitterName.Width = 80;
    lblSitterName.Text = "";
    foreach (Sitter sitter in sitterList) {
        if (sitter.SitterNumber == sitterNumber) {
            lblSitterName.Text = sitter.SitterName;
            break;
        }
    }
    lblSitterName.Left = 750;
    lblSitterName.Height = 12;
    lblSitterName.Top = 11;
    ledgerArray[row].Controls.Add(lblSitterName);

    // LEDGER ROW CHECKBOX.
    // The user approves a given appointment by checking its checkbox on the ledger row. Since approved
    // appointments create charge transactions for clients and pay entries for sitters, certain items
    // must be present on the ledger row in order for those entries to be made:
    // - There must be at least one client attached to the appointment.
    // - There must be a sitter attached to the appointment.
    // If any of the above is not true, the checkbox is not displayed on that row.

    if ((ledgerRow.FirstClientNumber != 0 || ledgerRow.SecondClientNumber != 0)
      && ledgerRow.SitterNumber > 0) {
        CheckBox chkApproval = new CheckBox();
        chkApproval.Name = "chkApproval";
        chkApproval.Text = "";
        chkApproval.Left = 840;
```

```
                chkApproval.Top = 9;
                chkApproval.Width = 12;
                chkApproval.TabStop = false;
                ledgerArray[row].Controls.Add(chkApproval);
            }

            // All controls have now been added to the ledger row.
            // The LedgerRowInfo item is added to the list of LedgerRowInfo items.
            ledgerRows.Add(ledgerRow);

            row++;
        }
    }
```

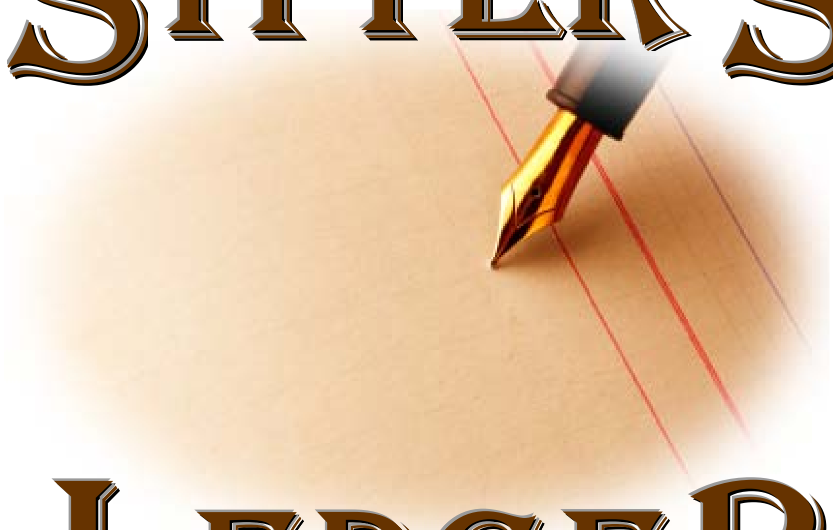
Fill History ListBoxes

Both the Billing page and the Payroll page contain a ListBox that shows transaction history, the former for clients and the latter for sitters. Since the format of the ListBox data is identical for both, a common method was created. The method is supplied a DataRowCollection with the transactions to be displayed, as well as the ListBox into which those transactions should be loaded. Some items—the date, transaction type, and transaction amount—are present in each ListBox row. The rest of the data on a given row depends on the type of transaction: Charge, Payment, or Adjustment. As a convenience to the user, after the ListBox has been loaded, it is positioned to display the most recent transactions.

```
private void FillHistoryListBox(DataRowCollection parDataRows, ListBox parListBox) {

    /* This method fills the billing history listbox or the payroll history listbox.
     * A single method is used for both types because the logic is virtually identical for both types.
     * The specific listbox that is filled, along with the dataset that is its source, are specified
     * as parameters to this method.
    */

    StringBuilder listboxItem;

    DateTime historyDateTime;
    decimal historyAmount;
    string historyDate;
    string historyTypeCode;
    string historyTypeText;
    string historyVariableText;
    int historyApptNumber;

    // Clear out the listbox to be loaded, and set its font to a fixed space font.
    parListBox.Items.Clear();
    parListBox.Font = new Font(FontFamily.GenericMonospace, 8.25F);

    // Instantiate the StringBuilder object that will be used to construct each row displayed
    // in the listbox.
    listboxItem = new StringBuilder();


    // The rows of the supplied dataset are now processed in order to create a string of the various
    // database columns, a string that will then be added as an entry in the listbox object. Each
    // column value is separated by a fixed number of spaces so each column lines up.

    foreach (DataRow dr in parDataRows) {
        // Clear out the StringBuilder object used to construct the data row string so it is empty
        // for each row.
        listboxItem.Remove(0, listboxItem.Length);
        // - Date value.
        if (DateTime.TryParse(dr["entryDate"].ToString(), out historyDateTime)) {
            historyDate = historyDateTime.ToShortDateString();
        }
        else {
            historyDate = string.Empty;
        }
        listboxItem.Append(string.Format("{0,-10}", historyDate));
        // - Entry type.
        historyTypeCode = dr["entryType"] != DBNull.Value ? dr["entryType"].ToString() : string.Empty;
        historyTypeText = string.Empty;
        switch (historyTypeCode) {
            case "A":
                historyTypeText = "Adjust.";
                break;
            case "C":
                historyTypeText = "Charge";
                break;
            case "P":
                historyTypeText = "Payment";
                break;
        }
        listboxItem.Append("  ");
        listboxItem.Append(string.Format("{0,-7}", historyTypeText));
        // - Entry amount.
        if (!decimal.TryParse(dr["entryAmount"].ToString(), out historyAmount)) {
            historyAmount = 0M;
        }
        historyAmount = historyTypeCode == "P" ? historyAmount * -1 : historyAmount;
        listboxItem.Append("  ");
        listboxItem.Append(string.Format("{0,7}", historyAmount.ToString("N")));
        // - Variable text.
        //   This will be the adjustment description for an adjustment transaction, an appointment type
        //   description and the child name for a charge transaction, or the check number for a payment
```

```
        //   transaction.
        historyVariableText = string.Empty;
        switch (historyTypeCode) {
            case "A":
                historyVariableText = dr["entryDesc"] != DBNull.Value ? dr["entryDesc"].ToString() :
string.Empty;
                break;
            case "C":
                if (int.TryParse(dr["apptNumber"].ToString(), out historyApptNumber)) {
                    historyVariableText = GetDataForAppt(historyApptNumber);
                }
                break;
            case "P":
                historyVariableText = "Check number: " + dr["entryCheckNumber"].ToString();
                break;
        }
        listboxItem.Append("   ");
        listboxItem.Append(string.Format("{0,-70}", historyVariableText));
        // Add as an item to the listbox the string that was built.
        parListBox.Items.Add(listboxItem.ToString());
    }

    // All items have been added to the history listbox.
    // Position the listbox so that the most recent items are those displayed.
    parListBox.TopIndex = parListBox.Items.Count - 1;
}
```

SITTER'S LEDGER



USER MANUAL

Contents

Installation

Installation of Sitter's Ledger requires a DVD drive and approximately 2MB of disk space.

To install Sitter's Ledger:

- Create a folder for holding the program files.
- Copy all files from the Sitter's Ledger program disc to the new folder.
- (Optional) Create a shortcut on the Windows desktop or in the Start menu to the SittersLedger.exe file.

Terminology

As you peruse this manual, it will help to know the meanings of some of the terms that are used.

Actual Start Time / Actual End Time

Most child care episodes start at a regular, given time (the Standard Start Time) and end at a regular time (the Standard End Time), points in time that do not vary. However, a child may arrive or leave earlier or later than the standard time. The Actual Start Time and Actual End Time are used for recording those variances.

Appointment

A single episode of care on a given day—either before school or after school—for a given child. An episode of care will be either before school or after school, so a given child can have at most two appointments on any given day.

Appointment Type

A single episode of care can occur either before school or after school. "Before School" and "After School" are the appointment types used in Sitter's Ledger.

Base Rate

The amount usually charged per child for a given appointment type. For example, Jane Smith typically is cared for after school twice a week. She is usually present from 2:45 – 5:00. The standard charge for each such occurrence is $7.00—the base rate. Using a base rate eliminates the need to specify an amount to be charged for each appointment; instead, this default charge amount is assumed. For those few occasions where Jane Smith stays either shorter or longer than the usual period of time, the base rate can be overridden.
For each client, there are separate before school and after school base rates.

Client

A customer of the child care business, responsible for paying for the child care.

Family
An entity that links individual children, and associates clients to children. For example, if Jane Smith and John Smith are part of one family, they would be assigned to single Sitter's Ledger family. One or two clients are assigned to the family as well. Two clients are used if two individuals are separately responsible for paying for portions of the family's child care.

Schedule
A specific before-school or after-school child care session. There can be one or more appointments for a given schedule instance. There can be at most two schedule instances on any given day: one before school and one after school. Each schedule instance is assigned one sitter.

Sitter
An individual who provides child care and is paid for doing so.

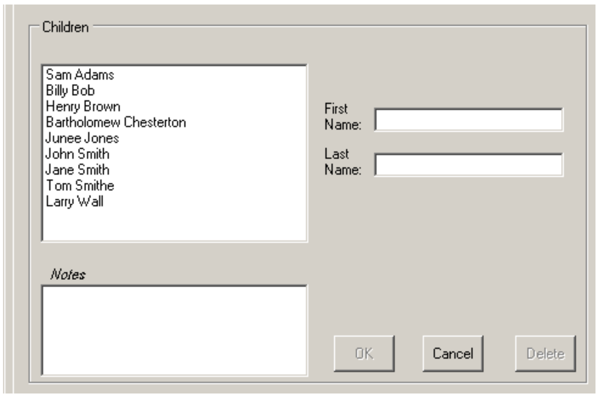
Standard Start Time / Standard End Time
The customary times at which a child arrives and leaves. Since most appointments will begin and end at unvarying times, using these standard values eliminates the need to record the arrival and departure times for every child every day. Sitter's Ledger does allow these values to be altered for those occasions for which a child may have arrived earlier or departed later than the normal times (see Actual Start Time / Actual End Time). There are separate Standard Start Time / End Time pairs for before school and after school. They are defined on the Sitter's Ledger Setup page.

Common Functionality

Some features of Sitter's Ledger are common to several functions. Those items are explained in this section.

Change Mode Indication
Several of the functions in Sitter's Ledger involve the adding of new items and the changing of existing items, both of which are done through a common window. To signal to the user that a given item is an existing one that is being changed—and is not a new one being added—the list from which the item being changed was selected is shaded.

For example, the images below are of the child maintenance utility. The image on the left shows the list of children as it appears when first displayed and when a new child is being entered. On the right, a child has been selected in the list, which Sitter's Ledger has shaded to signal that you are now in "change" mode.

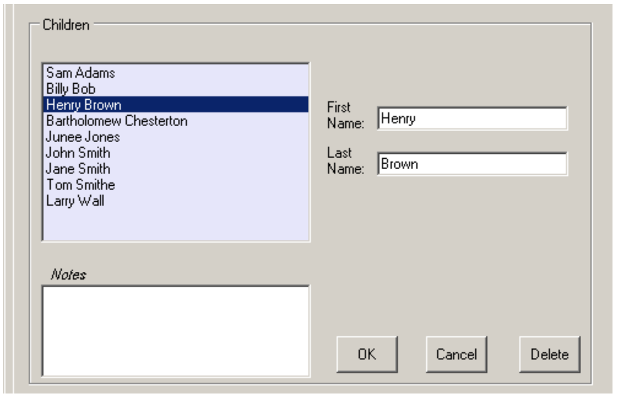

Children

Sam Adams
Billy Bob
Henry Brown
Bartholomew Chesterton
Junee Jones
John Smith
Jane Smith
Tom Smithe
Larry Wall

First Name:

Last Name:

Notes


Children

Sam Adams
Billy Bob
Henry Brown
Bartholomew Chesterton
Junee Jones
John Smith
Jane Smith
Tom Smithe
Larry Wall

First Name: Henry

Last Name: Brown

Notes

This change mode indication is used wherever a Sitter's Ledger utility is used to both add and change items.

Data Entry Shortcuts

Many of the data items keyed into boxes on Sitter's Ledger pages are of common types of information. For example, there are several places where time of day values are entered, several places where currency (dollar-and-cent) values are entered, and so on. As a convenience, for some types of data Sitter's Ledger provides shortcuts that make data entry quicker and easier than keying in the entire value would be. Those shortcuts are described below.

A shortcut is activated after you have entered a value and have left the box into which the value was entered. "Leaving" a data entry box means that you have pressed the Tab key to move to another field, that you have clicked on another data entry field on the screen, or that you have clicked on a button. If a value is entered that Sitter's Ledger does not recognize (for example, entering letters in a currency field), shortcuts will not be performed.

Currency Shortcuts

This shortcut eliminates the need for keying in the decimal point and the cents for even dollar amounts. For example, if you key in **5**, the shortcut will replace that value with **5.00**. A value keyed in with more than two digits after the decimal point will be rounded. (For example, **5.456** becomes **5.46**.)

Date Shortcuts

Valid dates in Sitter's Ledger consist of a month, day, and year with century, separated by slashes (for example, 2/8/2011). There are several shortcuts for date entry values:

- If only the month and day segments—separated by a slash—are entered, Sitter's Ledger will append the second slash and the year. For example, if **2/8** is entered, it will be replaced with **2/8/2011**.
- If a **T** (uppercase or lowercase) is entered, Sitter's Ledger will replace it with the current date.
- If a **Y** (uppercase or lowercase) is entered, Sitter's Ledger will replace it with yesterday's date.
- If a **T** (uppercase or lowercase) is entered with an addition or subtraction expression, Sitter's Ledger will replace it with the calculated date. For example, if **t-2** is entered, that expression will be replaced with the date that is two days prior to the current date.

Time Shortcuts

Valid times in Sitter’s Ledger consist of a hour and minute—separated by a colon—followed by a space and either AM or PM. The time shortcuts are:

- If the value keyed in ends with A or P (uppercase or lowercase), the letter M will be appended to the value. For example, **5:00 a** becomes **5:00 AM**.
- If the value ends with AM or with PM, and there is not a space just before the AM or PM, a space is inserted. For example, **5:00AM** becomes **5:00 AM**.
- If the value does not end with A, P, AM, or PM, Sitter’s Ledger will append an AM or PM based on whether the field is usually filled with an AM value or a PM value. For example, if a value of **5:00** is entered in the Before School Start Time field on the Setup page, it will be replaced with **5:00 AM**.

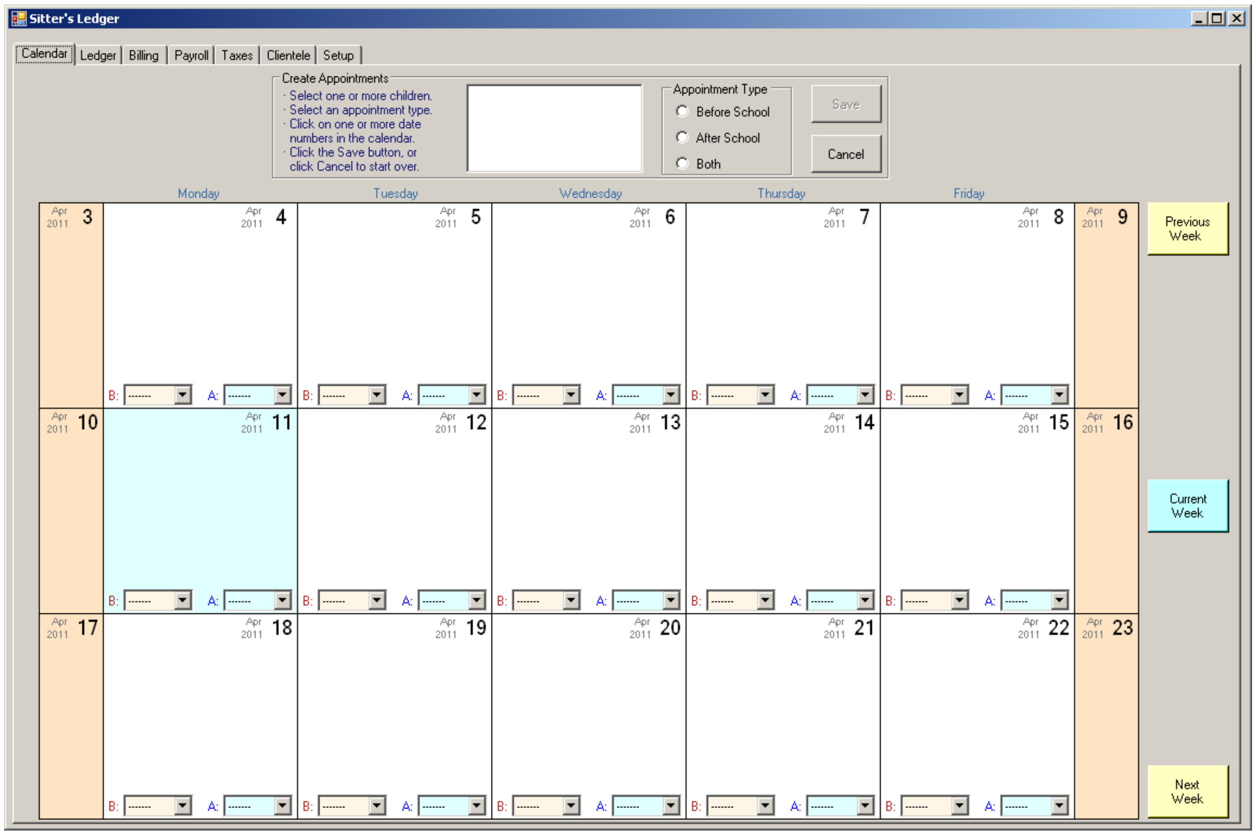
Phone Number Shortcuts

Phone numbers are stored in the Sitter’s Ledger database with their components separated by hyphens (e.g., 262-123-4567). Any value entered in a phone number field that contains seven or ten digits—or eleven digits beginning with the number 1—will be reformatted into the Sitter’s Ledger phone number format. Examples:

- If you key in 1234567, it will be replaced with **123-4567**.
- If you key in **123 4567**, it will be replaced with **123-4567**.
- If you key in **2223334444**, it will be replaced with **222-333-4444**.
- If you key in **15556667777**, it will be replaced with **1-555-666-7777**.
- If you key in **(414) 888-9999**, it will be replaced with **414-888-9999**.

Using Sitter’s Ledger

When Sitter’s Ledger is launched, the Calendar window shown below is displayed.

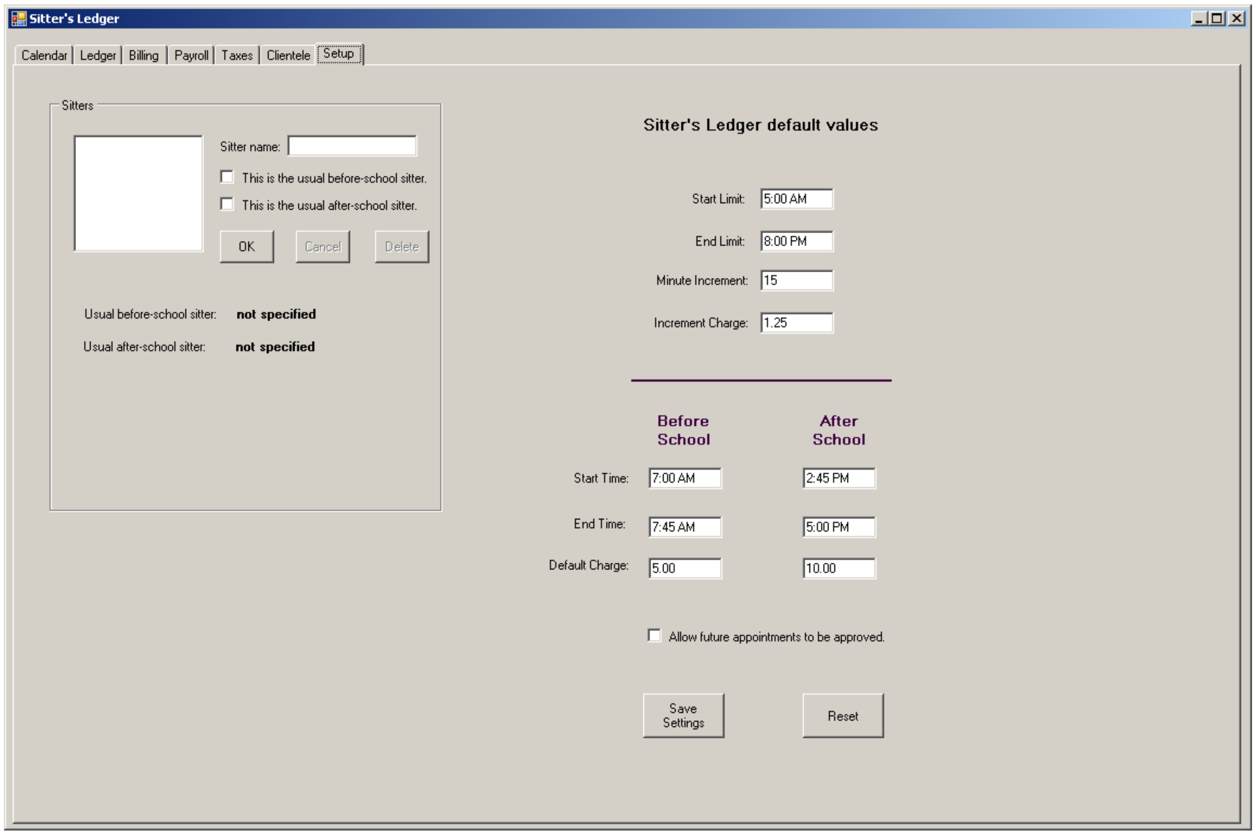


Accessing the various functions of Sitter's Ledger is done via the series of page tabs located in the upper left area of the window, enlarged in the image below:

Because the primary functions of Sitter's Ledger use some key pieces of information (the children being cared for, the sitters providing care, etc.), the setup of that information is addressed first.

The SETUP Page

Open the Setup page by clicking on the Setup tab. The Setup page includes a Sitter maintenance section on the left, and a Default Value section on the right:



Sitter's Ledger

Calendar | Ledger | Billing | Payroll | Taxes | Clientele | Setup

Sitters

Sitter name:

☐ This is the usual before-school sitter.

☐ This is the usual after-school sitter.

OK Cancel Delete

Usual before-school sitter: **not specified**

Usual after-school sitter: **not specified**

Sitter's Ledger default values

Start Limit: 5:00 AM

End Limit: 8:00 PM

Minute Increment: 15

Increment Charge: 1.25

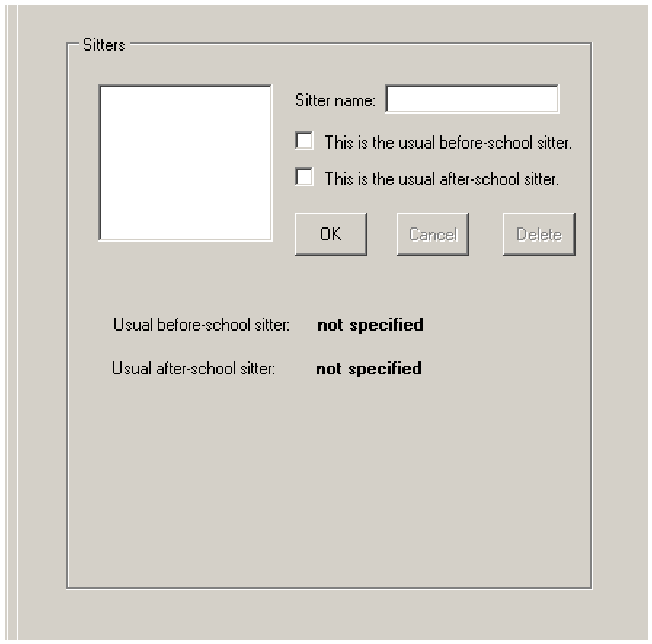
	Before School	After School
Start Time:	7:00 AM	2:45 PM
End Time:	7:45 AM	5:00 PM
Default Charge:	5.00	10.00

☐ Allow future appointments to be approved.

Save Settings Reset

Sitter maintenance

The section of the page into which Sitter information is entered is shown below:


Sitters
Sitter name:
This is the usual before-school sitter.
This is the usual after-school sitter.
OK
Cancel
Delete

Usual before-school sitter: **not specified**

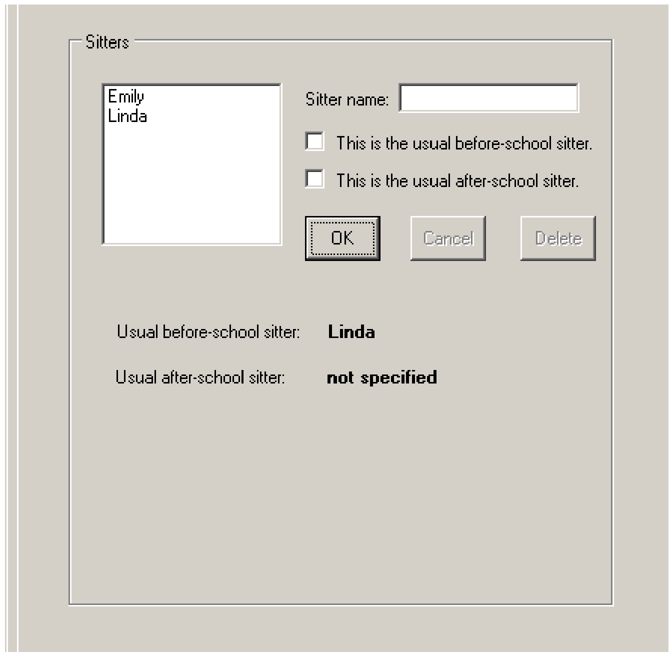
Usual after-school sitter: **not specified**

Add a new sitter
Key the sitter's name into the box to the right of the words "Sitter name:". (If the flashing curser is not in the sitter name box, click on it with the mouse to position the cursor there.)

If this sitter typically will be on duty before school, click on the *This is the usual before-school sitter* checkbox; otherwise, leave the checkbox blank. If the sitter typically will be on duty after school, click on the *This is the usual after-school sitter* checkbox. These checkboxes allow Sitter's Ledger to automatically schedule the sitters who will usually be on duty, so that you don't have to make those selections for every day that child care occurs. If you use the usual sitter feature, be aware that on the scheduling calendar the usual sitter can be overridden with any other sitter.

To save the information for this sitter, click on the OK button. Click on the Cancel button to abandon the items you have entered. When you click the OK button, the name of the sitter will be listed in the large box at the left. If you have selected the sitter as a usual before-school or after-school sitter, that sitter's name will be shown in the Sitters window in place of the words "not specified".

In the sample illustration below, two sitters have been added. One of them has been designated as the usual before-school sitter, and her name is listed as such.


Sitters
Emily
Linda
Sitter name:
This is the usual before-school sitter.
This is the usual after-school sitter.
OK
Cancel
Delete

Usual before-school sitter: **Linda**

Usual after-school sitter: **not specified**

Modify sitter Information
To change a sitter's information (for example, the name or the usual sitter designation), click on the sitter's name in the list. The name will be displayed in the *Sitter name:* box, and a usual sitter checkbox will be checked if applicable.

You can change the sitter's name, and change usual sitter designations. If you uncheck a usual sitter checkbox, that sitter will no longer show up on the calendar as the usual sitter. (The sitter name on the calendar will not change, however, on any day that has any appointments already scheduled.) If you check a usual sitter checkbox that was previously checked for another sitter, the designation is removed from the other sitter and is assigned to the sitter that you are changing.

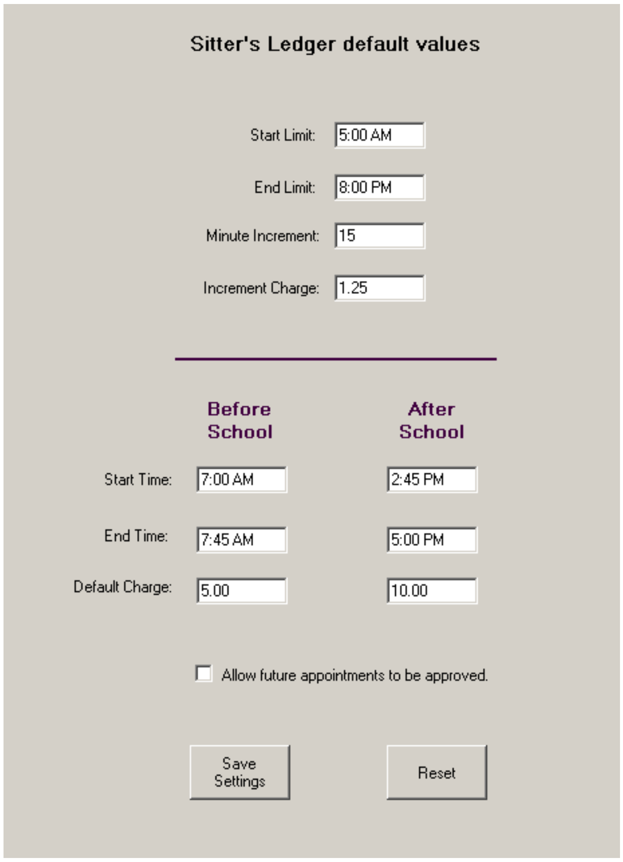
Click the OK button to save the changes; click the Cancel button to abandon the changes.

Delete a sitter
To delete a sitter, click on the sitter's name in the list. The name will be displayed in the *Sitter name:* box. Click on the Delete button.

Note: Sitter's Ledger will not permit a sitter who has existing pay records (which are created on the Payroll page) to be deleted. A message will be displayed if you attempt to do so.

Default value maintenance

Several values that are used repeatedly through the pages of Sitter's Ledger can be given default values, eliminating the need for the user to have to enter these values. The section of the Setup page in which these values are entered is shown below:



Sitter's Ledger default values

Start Limit:	5:00 AM
End Limit:	8:00 PM
Minute Increment:	15
Increment Charge:	1.25

	Before School	After School
Start Time:	7:00 AM	2:45 PM
End Time:	7:45 AM	5:00 PM
Default Charge:	5.00	10.00

☐ Allow future appointments to be approved.

Note that the program supplies best-practice default values. These can be changed as needed as the nature of the child care business changes.

Descriptions of the individual items in this section follow. If you change values and want to keep them, click the Save Settings button. If you have changed values and decide to abandon those changes, click the Reset button.

Start Limit
This is the earliest time of day that a given child care appointment can start. When entering or changing an appointment, Sitter's Ledger will inform you if you have selected a start time that is earlier than this value.

End Limit
This is the latest time of day that a given child care appointment can end. When entering or changing an appointment, Sitter's Ledger will inform you if you have selected an end time that is later than this value.

Minute Increment
If a child is at day care for a period that is shorter or is longer than scheduled, it is typical that the amount charged to the customer be modified accordingly. This value indicates how many minutes must elapse before the amount charged to the customer changes, and is used in conjunction with the Increment Charge. These values simply allow Sitter's Ledger to make those alterations automatically, and can be overridden when a charge is actually reviewed and processed.

Increment Charge
This is the amount by which a customer's charge for a given day care appointment will be reduced or increased for each Minute Increment that has elapsed for that appointment. That calculation is automatically done by Sitter's Ledger, but the result can be overridden if necessary.

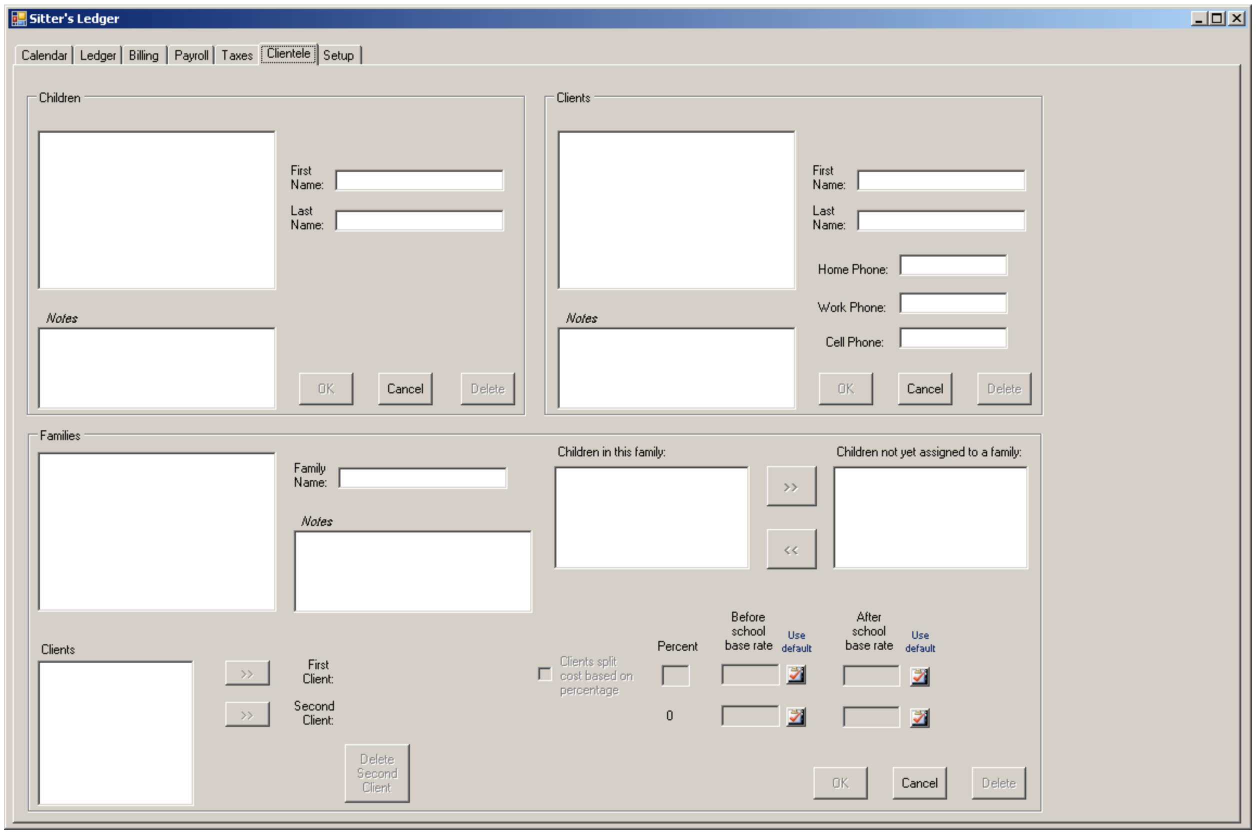
Start Time / End Time
Most day care appointments start and end at the same time for each child on any given day. These fields allow the typical start and end times—for both the Before School period and the After School period—to be defined. These will be the values used when an appointment is created, but can be overridden for any given appointment.

Default Charge
These values are used to specify the default amount charged per child for each Before School and each After School period. While most customers will be charged these amounts, Sitter's Ledger allows the amounts to be overridden by client if special rates are agreed upon (for example, if a customer is given a price break for multiple children).

Allow future appointments to be approved
By default, Sitter's Ledger will not allow you to charge for an appointment the end time of which is later than the current time. This is because you typically will not know how long the child will end up being on site until the child is picked up by his or her parent or guardian. If there is an occasion where the ability to approve an appointment whose end time is in the future, click on this checkbox to enable that capability.

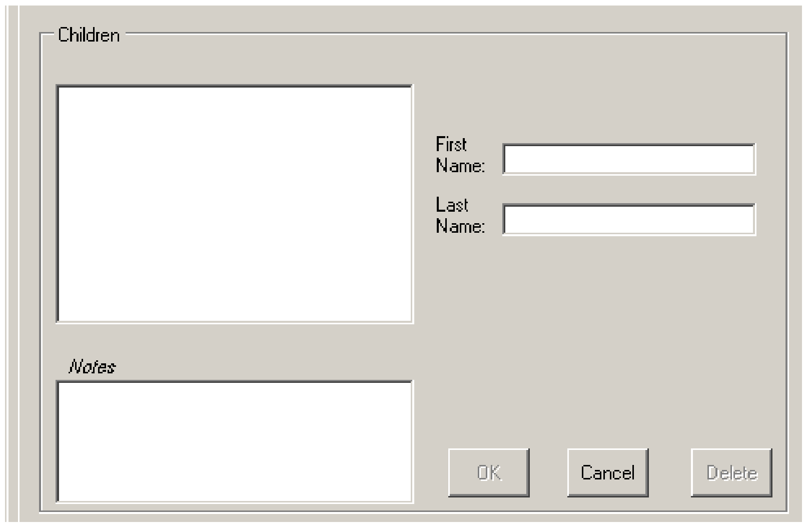
The CLIENTELE Page

Open the Clientele page by clicking on the Clientele tab. The Clientele page has three sections: one for maintaining the children, one for maintaining clients (the customers), and one for maintaining the families (which link children and clients):



Child maintenance

The section of the page into which Child information is entered is shown below:



Add a new child
Key the child's first name and last name into the boxes provided. An optional note can be recorded for each child by typing that information into the *Notes* box. (This box can be used, for example, to record any special needs the child may have.)

When done, click the OK button. The child's name will be added to the list of children in the large box at the top left area of this section. To abandon the entry of the child information, click the Cancel button.

Modify child information
To change a child's information, click on the child's name in the list. The corresponding information will be placed in the same field used for entering a new child.

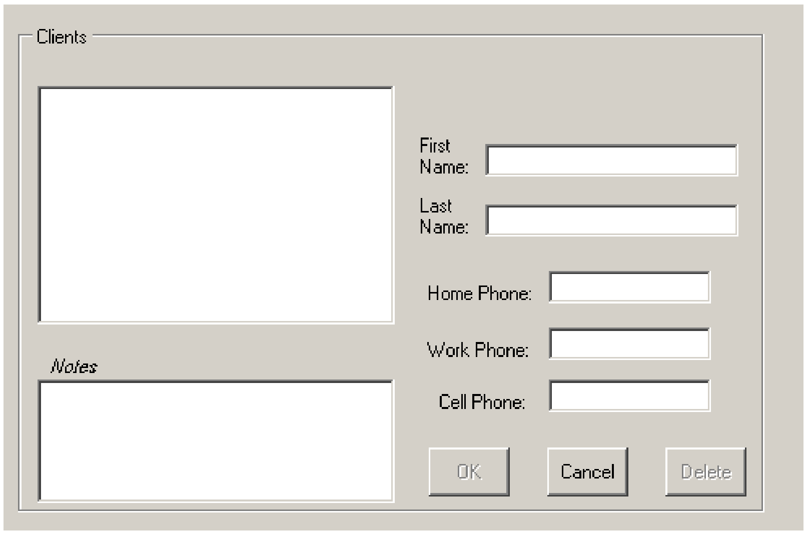
Click the OK button to save the changes; click the Cancel button to abandon the changes.

Delete a child
To delete a child, click on the child's name in the list. The corresponding information will be placed in the same field used for entering a new child. Click the delete button to remove the child from the database.

Note: Sitter's Ledger will not permit a child who has existing appointments to be deleted. A message will be displayed if you attempt to do so.

Client maintenance

The section of the page into which Client information is entered is shown below:



Add a new client
Key the client's first name and last name into the boxes provided. The *Notes* box is an area for recording notes, and is optional, as are the phone number items.

When done, click the OK button. The client's name will be added to the list of clients in the large box at the top left area of this section. To abandon the entry of the client information, click the Cancel button.

Modify client information
To change a client's information, click on the client's name in the list. The corresponding information will be placed in the same field used for entering a new client.

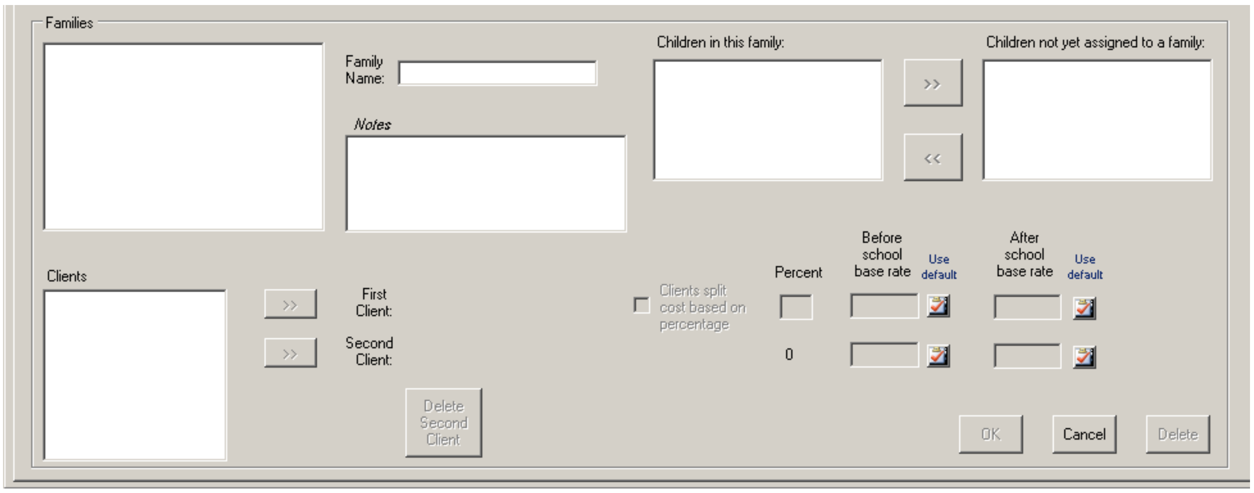
Click the OK button to save the changes; click the Cancel button to abandon the changes.

Delete a client
To delete a client, click on the client's name in the list. The corresponding information will be placed in the same field used for entering a new client. Click the delete button to remove the client from the database.

Note: Sitter's Ledger will not permit a client who has existing transactions to be deleted. A message will be displayed if you attempt to do so.

Family maintenance

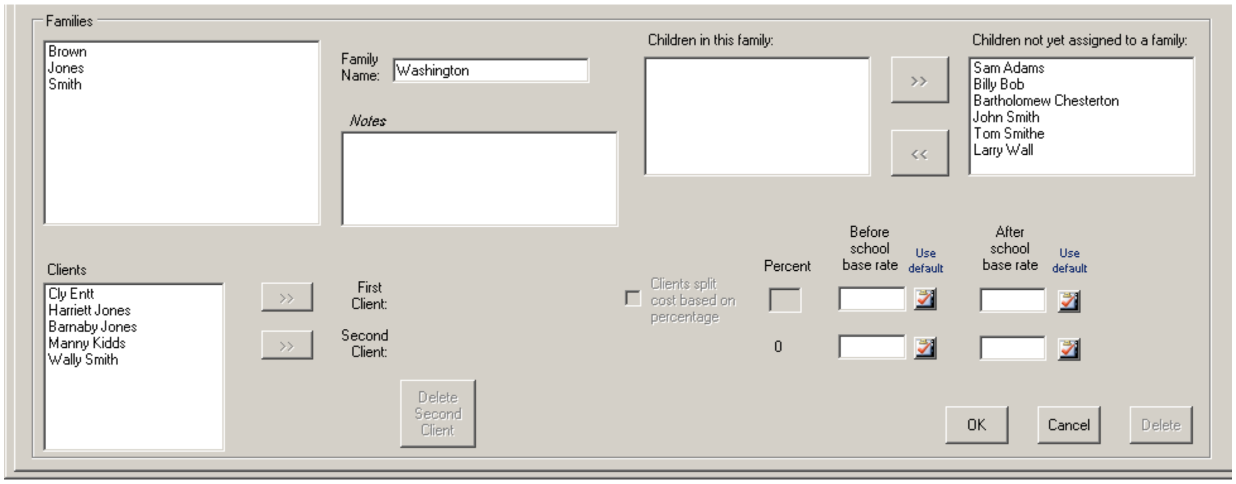
The section of the page into which Family information is entered is shown below:



Since families in Sitter's Ledger consist of children and clients, it is recommended that family maintenance be done after all children and clients have been entered.

Add a new family
Key the family's name into the box labeled *Family Name:*. This is typically just a last name—like Smith or Jones—and is typically the name by which the children are known to the sitter. The *Notes* box is an area for recording notes, and is not required.

The remaining fields will be explained using the example shown below, where the family name Washington has been entered.



Families:
Brown
Jones
Smith

Family Name: Washington

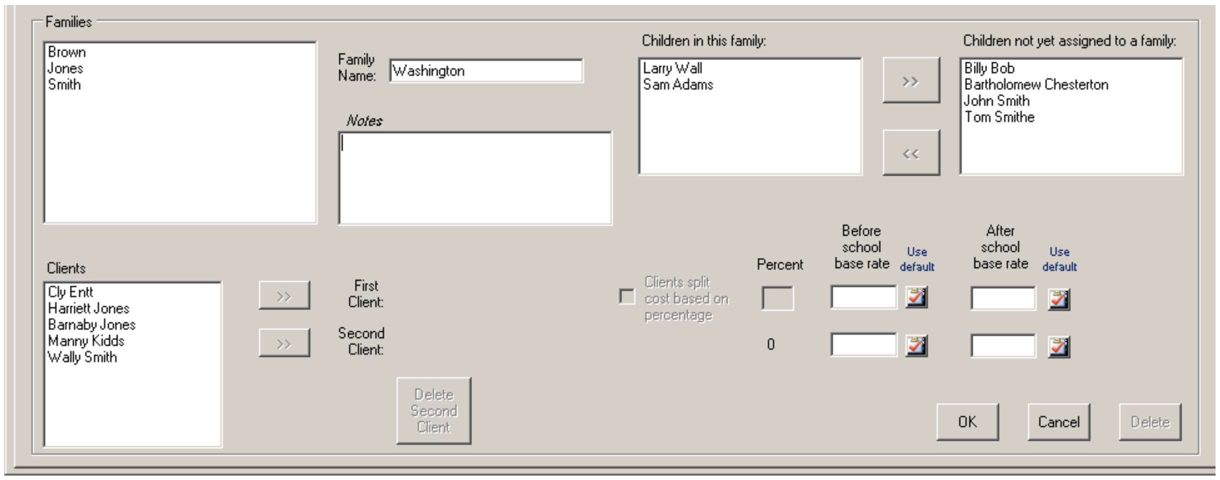
Children not yet assigned to a family:
Sam Adams
Billy Bob
Bartholomew Chesterton
John Smith
Tom Smithe
Larry Wall

Clients:
Cly Entt
Harriett Jones
Barnaby Jones
Manny Kidds
Wally Smith

Add children to the family
To add children to the family, click on one or more names in the *Children not yet assigned to a family* list, then click the << button next to that list. (A name can be deselected by clicking on it again.) The names will be removed from that list and placed into the *Children in this family* list.

If a mistake has been made, a child can be removed from the family by clicking on the child's name in the *Children in this family* list, then clicking the >> button next to that list.

In the example below, two children have been added to the family.



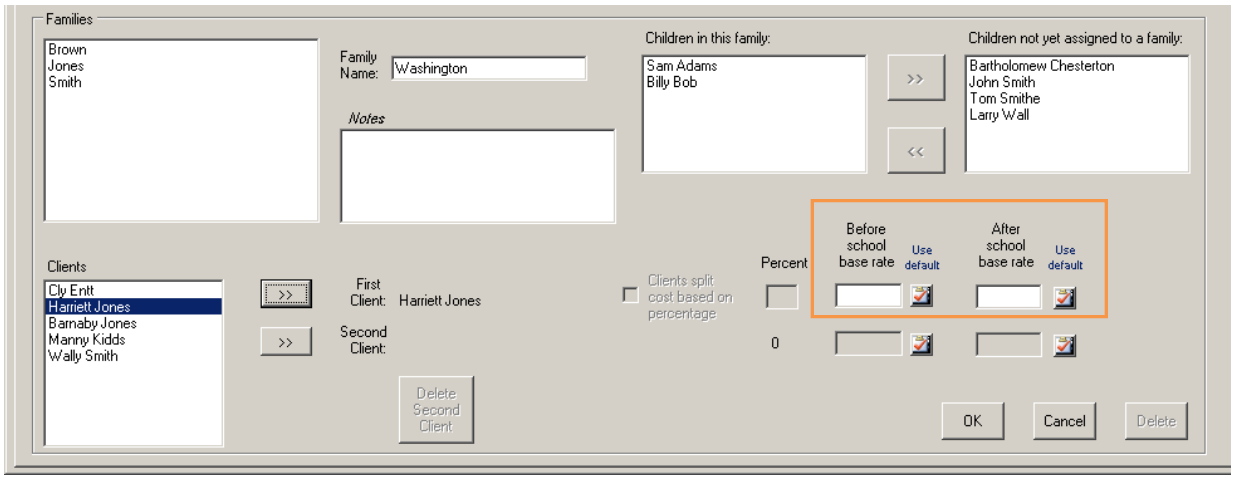
Add clients to the family

To add a client to the family, click on a client name in the Clients list in the Families section of the page, then click the >> by the First Client label to add the client as the first (or only) client. Add a second client by clicking on another name, then clicking on the >> button next to the Second client label. You cannot assign the same client name to both the First Client and the Second Client within a given family.

Either client can be changed by clicking on a different client name, then pressing the appropriate >> button. If there is a second client, it can be deleted (making the family a one-client family instead of a two-client family) by clicking on the Delete Second Client button. (This button is active only if there are two clients assigned to the family.)

Record the clients base charge rates

In the example below, the family has one client. The rates that this client normally will be charged for each child before school and after school are entered in the *Before school base rate* and *After school base rate* fields, highlighted in the example. This can be any dollar-and-cent amount. To use the default values specified on the Setup page, click the checkmark button next to the base rate box.


Families

Brown
Jones
Smith

Family Name: Washington

Notes

Children in this family:

Sam Adams
Billy Bob

Children not yet assigned to a family:

Bartholomew Chesterton
John Smith
Tom Smithe
Larry Wall

Clients

Cly Entt
Harriett Jones
Barnaby Jones
Manny Kidds
Wally Smith

First Client: Harriett Jones

Second Client:

Clients split cost based on percentage

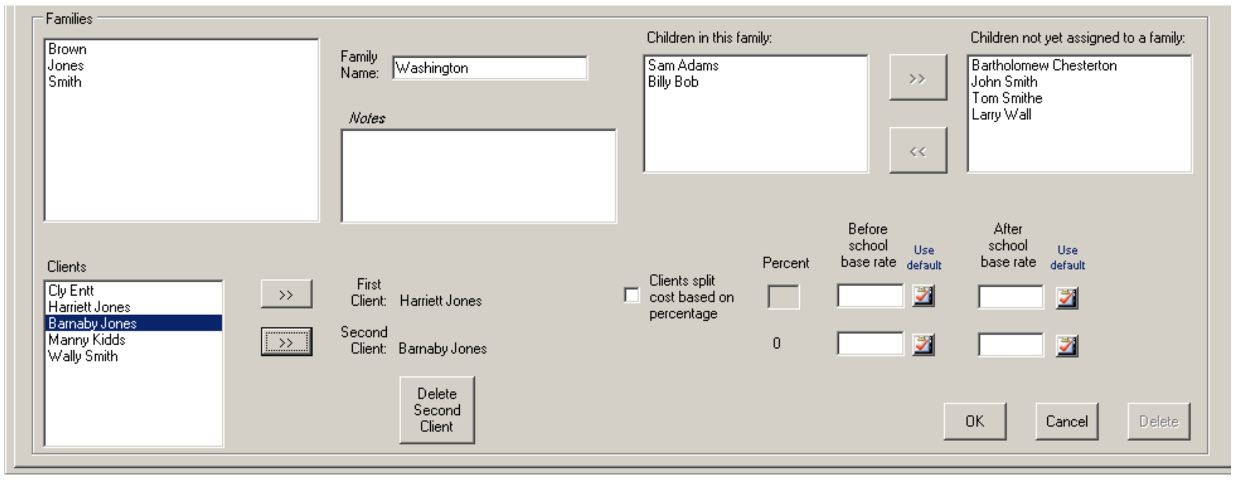
Percent

Before school base rate

After school base rate

0

In the next example, the family has two clients. Note that the *Clients split cost based on percentage* checkbox is enabled, as are the second client's base rate boxes. If the clients will not be paying based on a percentage split, leave the checkbox unchecked, and enter the second client's base rates in the same way that the first client's were.


Families

Brown
Jones
Smith

Family Name: Washington

Notes

Children in this family:

Sam Adams
Billy Bob

Children not yet assigned to a family:

Bartholomew Chesterton
John Smith
Tom Smithe
Larry Wall

Clients

Cly Entt
Harriett Jones
Barnaby Jones
Manny Kidds
Wally Smith

First Client: Harriett Jones

Second Client: Barnaby Jones

Clients split cost based on percentage

Percent

Before school base rate

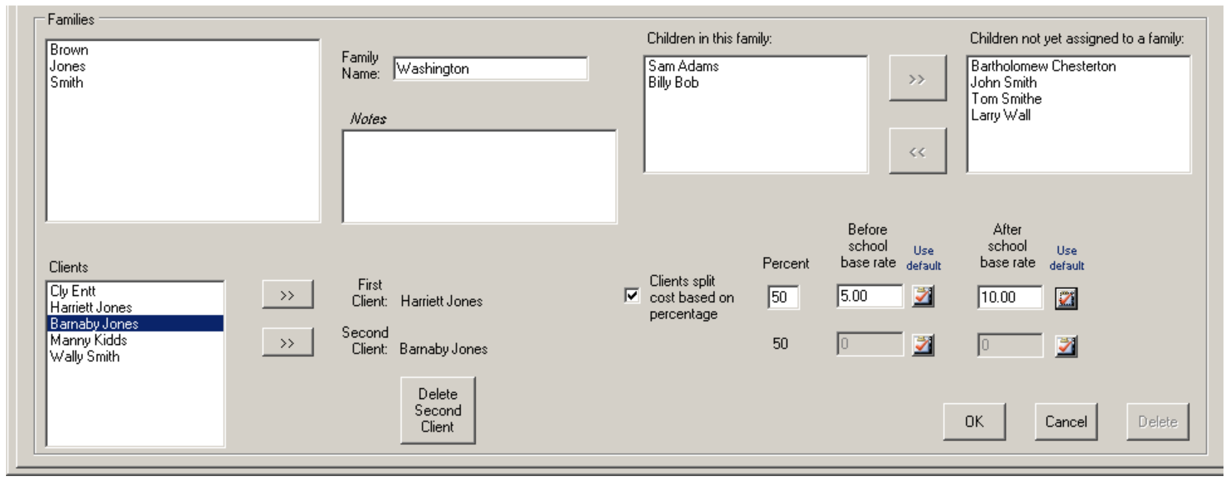
After school base rate

0

For a family whose clients split the cost based on a percentage, check the *Clients split cost based on percentage* checkbox. Because the second client's cost is calculated automatically, the base rate fields for the second client will be disabled when the percentage checkbox is checked.

Enter in the *Percent* box the percentage that the first client will pay. The second client's percentage will be automatically calculated. The total base rates per child should be entered in the first client's base rate fields. For example, if the amount to be charged for the family's children for before school appointments is five dollars, that is the figure that should be entered in the Before School Base Rate box. The amounts for each client based on the percentage will not be shown on this page, but will be displayed on the Ledger page during the appointment approval process.

The example below shows a family whose clients split the cost evenly. Each client will be responsible for $2.50 per child for before school appointments and $5.00 per child for after school appointments.



Families:
Brown
Jones
Smith

Family Name: Washington

Notes

Children in this family:
Sam Adams
Billy Bob

Children not yet assigned to a family:
Bartholomew Chesterton
John Smith
Tom Smithe
Larry Wall

Clients:
Cly Entt
Harriett Jones
Barnaby Jones
Manny Kidds
Wally Smith

First Client: Harriett Jones
Second Client: Barnaby Jones

[✓] Clients split cost based on percentage

Percent	Before school base rate	After school base rate
50	5.00	10.00
50	0	0

After entering all data for the family, click the OK button. Click the Cancel button to abandon entry of the family.

Modify family information
To change a family's information, click on the family's name in the list. The corresponding information will be placed in the same field used for entering a new family. Make changes per the explanations under *Add a new family* above.

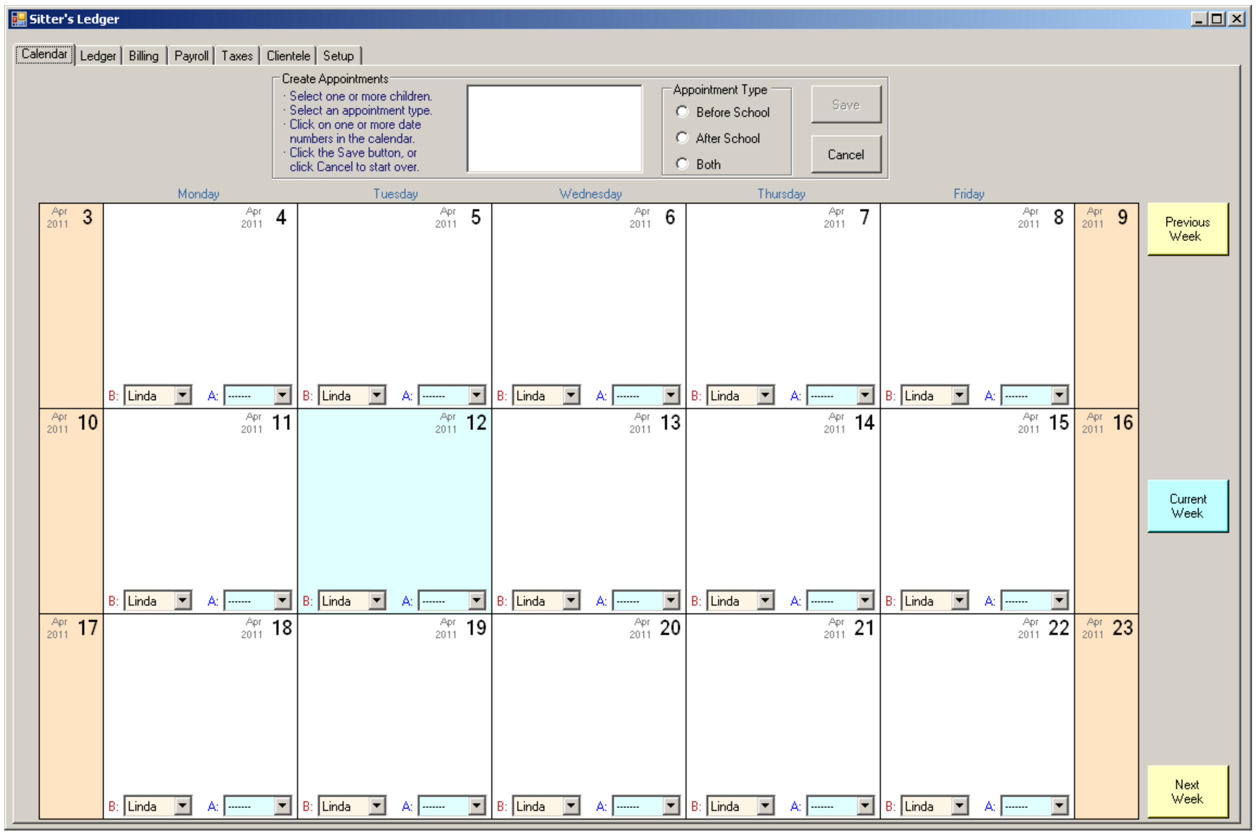
Click the OK button to save the changes; click the Cancel button to abandon the changes.

Delete a family
To delete a family, click on the family's name in the list. The corresponding information will be placed in the same field used for entering a new family. Click the delete button to remove the family from the database.

Note: Sitter's Ledger will not permit the deletion of a family to which children have been assigned. A message will be displayed if you attempt to do so.

The CALENDAR Page

Open the Calendar page by clicking on the Calendar tab. The Calendar page shows three weeks at a time:



Create Appointments
- Select one or more children.
- Select an appointment type.
- Click on one or more date numbers in the calendar.
- Click the Save button, or click Cancel to start over.

	Monday	Tuesday	Wednesday	Thursday	Friday	
Apr 2011 3	Apr 2011 4 B: Linda A: ------	Apr 2011 5 B: Linda A: ------	Apr 2011 6 B: Linda A: ------	Apr 2011 7 B: Linda A: ------	Apr 2011 8 B: Linda A: ------	Apr 2011 9
Apr 2011 10	Apr 2011 11 B: Linda A: ------	Apr 2011 12 B: Linda A: ------	Apr 2011 13 B: Linda A: ------	Apr 2011 14 B: Linda A: ------	Apr 2011 15 B: Linda A: ------	Apr 2011 16
Apr 2011 17	Apr 2011 18 B: Linda A: ------	Apr 2011 19 B: Linda A: ------	Apr 2011 20 B: Linda A: ------	Apr 2011 21 B: Linda A: ------	Apr 2011 22 B: Linda A: ------	Apr 2011 23

When Sitter's Ledger is launched, the current week is shown in the middle row of the calendar, and the prior week and next week are shown above and below the current week, respectively. The current date is always shaded in light blue. Weekend date cells are narrower than weekday cells because appointments cannot be scheduled on weekends.

Child care appointments and sitter schedules are entered on this page. The sitter schedules are specified in the drop-down lists at the bottom of each weekday page. In the preceding sample image, the before school sitter (preceded by **B:**) is filled in because a usual before school sitter was specified on the Setup page. The after school sitter list (preceded by **A:**) does not show a name because a usual after school sitter was not specified on the Setup page.

Changing the calendar period

To change the time period that is displayed, click the Previous Week button to go back one week; click the Next Week button to move forward one week. The Current Week button will return the calendar to the initial time period, with the current week in the center row.

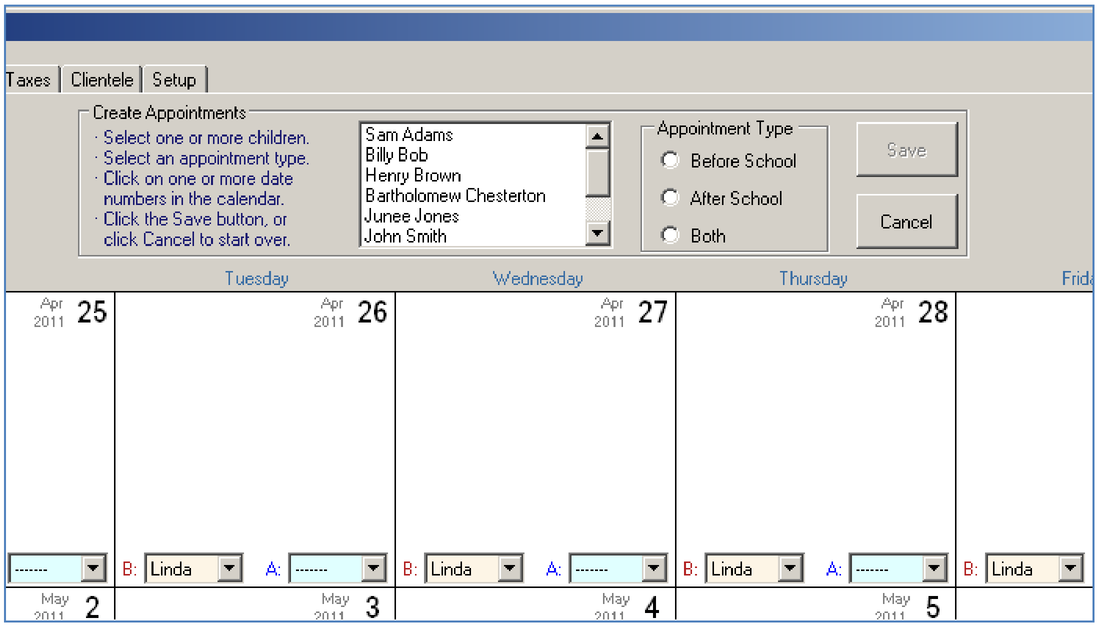
Creating appointments

Appointment creation is done by making three selections:

- The child or children.
- The appointment type (before school, after school, or both).
- The date or dates.

Appointment times are automatically given the default start and end times that have been entered on the Sitter's Ledger Setup page. Modifying those times will be discussed later in this section.

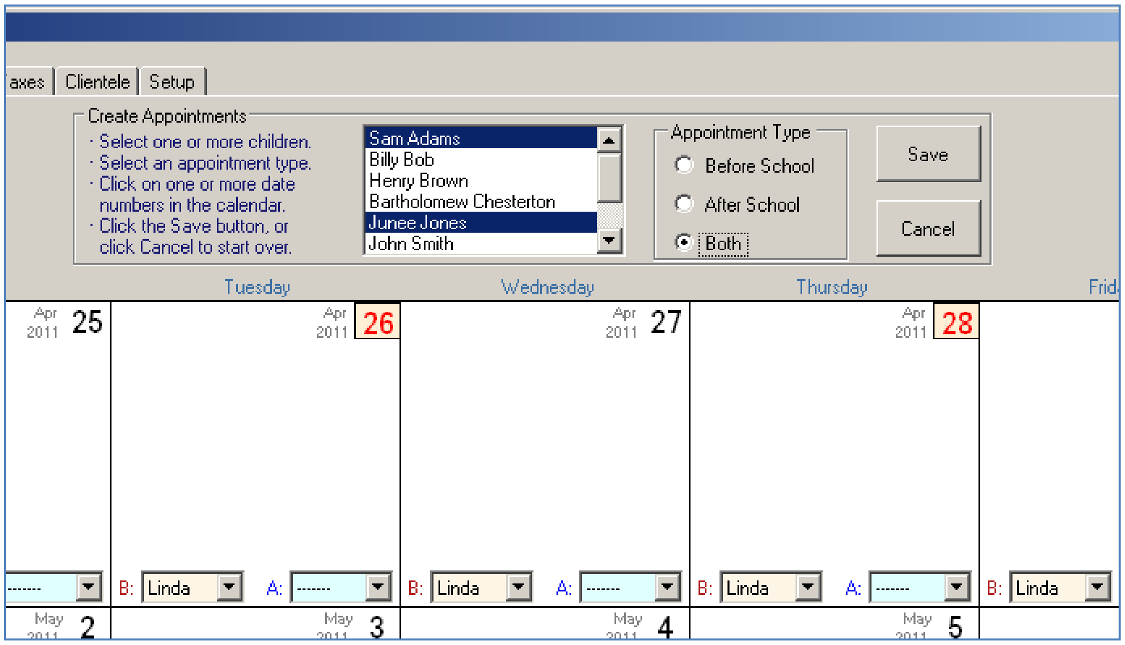
The image below—enlarged from the Calendar page—shows the appointment instruction list, the list of children, and the appointment type options.



To create appointments:

- Click on the name or names of the child or children for whom you want to create appointments. (Clicking again a name that is already highlighted will deselect that child.)
- Click on one of the appointment types.
- Click on the number or numbers of the day or days on which the appointments should be made. You must click on the number itself when selecting dates. When you click on a date, it is highlighted by changing its color. Click again on a highlighted date number to deselect it.

In the example below, two children, the "both" appointment type, and two dates have been selected.



axes | Clientele | Setup

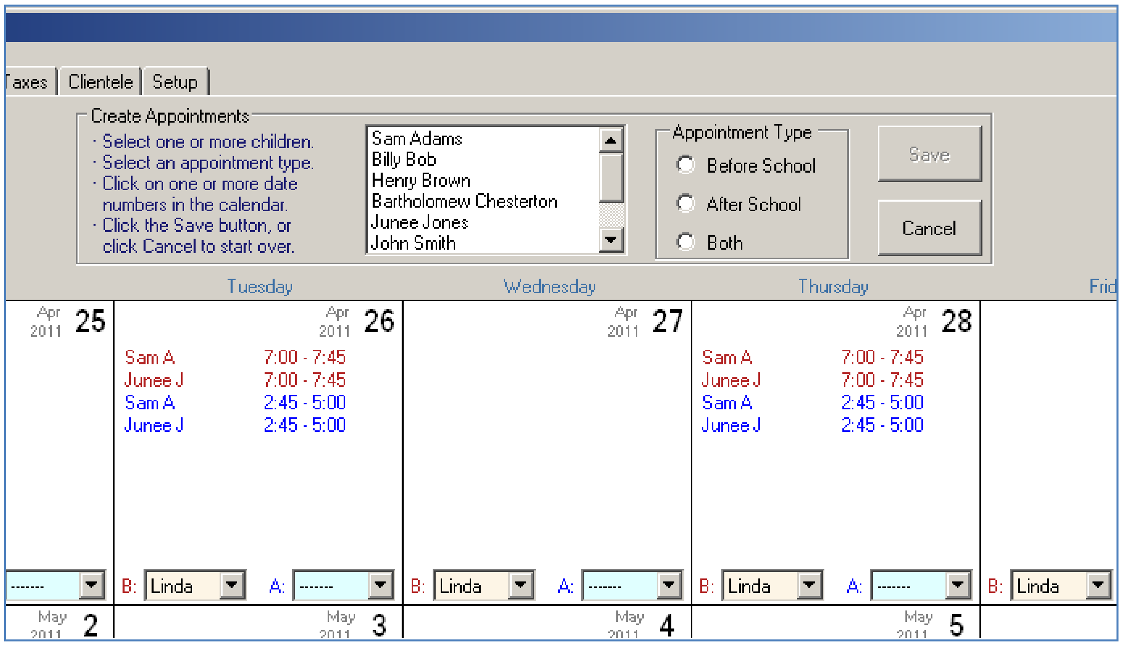
Create Appointments
- Select one or more children.
- Select an appointment type.
- Click on one or more date numbers in the calendar.
- Click the Save button, or click Cancel to start over.

Sam Adams
Billy Bob
Henry Brown
Bartholomew Chesterton
Junee Jones
John Smith

Appointment Type
Before School
After School
[✓ Both]

To complete the appointment creation process, click on the Save button. Click the Cancel button to abandon the appointment creation process. (Note that the calendar navigation buttons are disabled while you make appointments. After you have clicked the OK or Cancel buttons, those buttons become enabled again.)

The image below shows the appointments, based on the selections made , now on the calendar. The child's first name and last initial, along with the appointment period, are listed. For easy identification, before school appointments are shown in red and after school appointments are shown in blue.



Taxes | Clientele | Setup

Create Appointments
- Select one or more children.
- Select an appointment type.
- Click on one or more date numbers in the calendar.
- Click the Save button, or click Cancel to start over.

Sam Adams
Billy Bob
Henry Brown
Bartholomew Chesterton
Junee Jones
John Smith

Appointment Type
Before School
After School
Both

Apr 26, 2011:

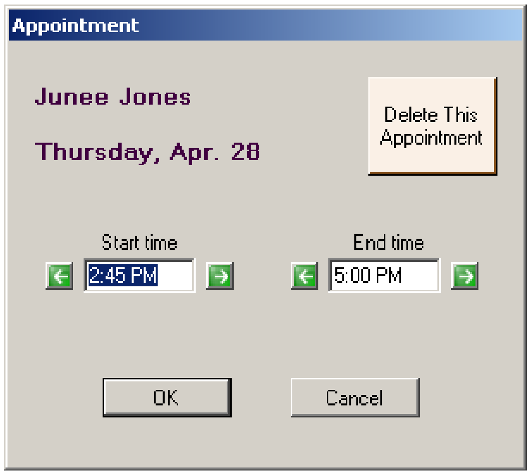
Sam A	7:00 - 7:45
Junee J	7:00 - 7:45
Sam A	2:45 - 5:00
Junee J	2:45 - 5:00

Apr 28, 2011:

Sam A	7:00 - 7:45
Junee J	7:00 - 7:45
Sam A	2:45 - 5:00
Junee J	2:45 - 5:00

Scheduling the sitters
Although appointments can be made without specifying the sitter, they cannot be charged for until a sitter has been specified for the date and appointment type on which they occur. Referring to the example in this section, you will see that the default before school sitter is shown at the bottom of each day's cell, but no after school sitter is. Select a sitter by clicking the down arrow on the drop-down lists, and clicking the name of the sitter who is scheduled to sit on a given date for a given appointment type. Even if a default sitter is listed, it can be changed through the same process.

Changing and deleting appointments
Sometimes a child's actual arrival or departure time deviates from the default. If that deviation is only a few minutes, it may not be a concern, and you probably won't change the amount charged to the customer for that appointment. But if the appointment duration deviates substantially, Sitter's Ledger provides a way to change the start and end times for an appointment. An appointment can be deleted altogether as well. To change or delete an appointment, click on its entry on the calendar. A small Appointment window will be displayed:


Appointment

Junee Jones

Thursday, Apr. 28

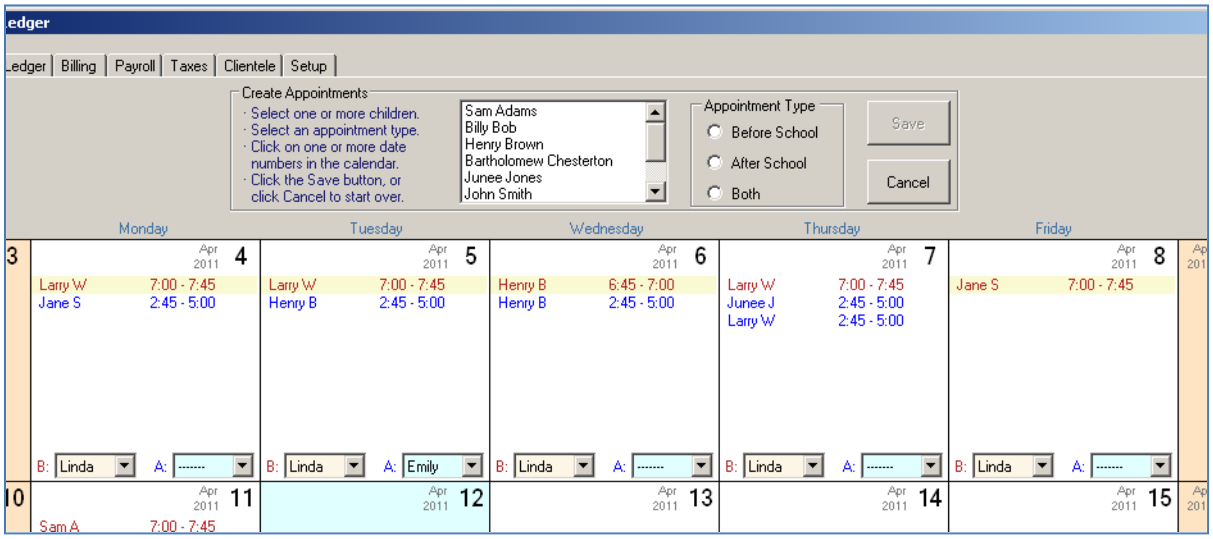
Start time: 2:45 PM

End time: 5:00 PM

To delete the appointment, click on the Delete This Appointment button. Start and end times can be keyed into the fields, or can be changed using the left and right arrow buttons. These buttons alter the time by the number of minutes specified in the *Minute Increment* item on the Sitter's Ledger Setup page. Save the modified appointment by clicking on the OK button. Click the Cancel button to abandon the changes.

Approved Appointments
Sitter's Ledger will not permit the modification or deletion of an approved appointment (which is one for which a charge has been generated). As an aid to the user, approved appointments are highlighted on the calendar.

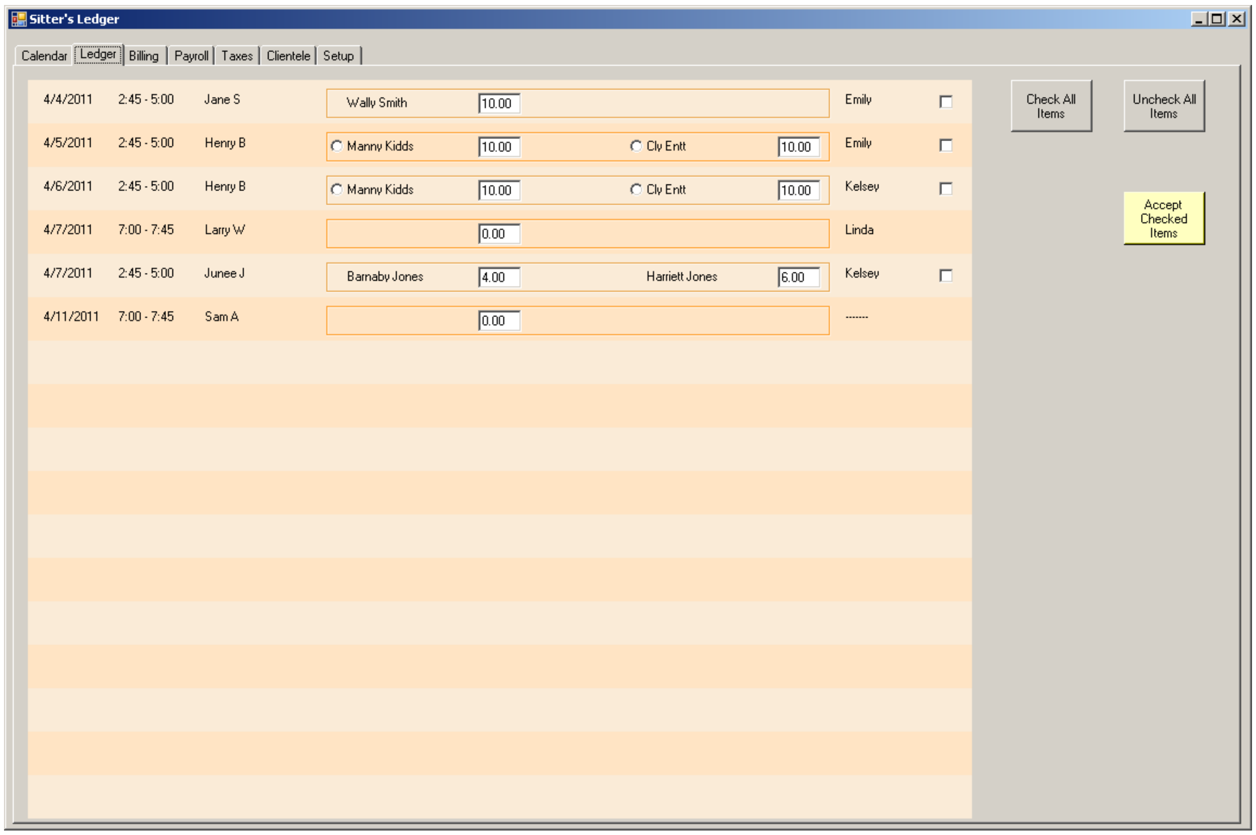
In the partial calendar image below, all the before school appointments are highlighted, indicating that those appointments have been approved and cannot be modified or deleted.


edger
Ledger | Billing | Payroll | Taxes | Clientele | Setup
Create Appointments
· Select one or more children.
· Select an appointment type.
· Click on one or more date numbers in the calendar.
· Click the Save button, or click Cancel to start over.
Sam Adams
Billy Bob
Henry Brown
Bartholomew Chesterton
Junee Jones
John Smith
Appointment Type
Before School
After School
Both
Save
Cancel

Monday	Tuesday	Wednesday	Thursday	Friday
Apr 2011 4	Apr 2011 5	Apr 2011 6	Apr 2011 7	Apr 2011 8
Larry W 7:00 - 7:45	Larry W 7:00 - 7:45	Henry B 6:45 - 7:00	Larry W 7:00 - 7:45	Jane S 7:00 - 7:45
Jane S 2:45 - 5:00	Henry B 2:45 - 5:00	Henry B 2:45 - 5:00	Junee J 2:45 - 5:00	
			Larry W 2:45 - 5:00	
B: Linda A: ------	B: Linda A: Emily	B: Linda A: ------	B: Linda A: ------	B: Linda A: ------
Apr 2011 11	Apr 2011 12	Apr 2011 13	Apr 2011 14	Apr 2011 15
Sam A 7:00 - 7:45				

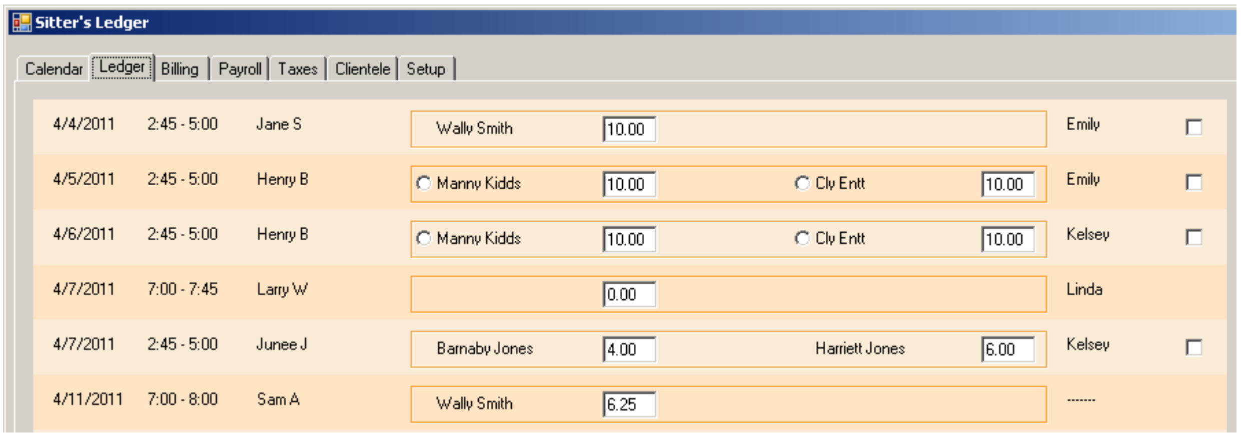
The LEDGER Page

Open the Ledger page by clicking on the Ledger tab. The Ledger page is where appointments are approved, and is intended to be used regularly so you don't get too far behind in billing customers and paying sitters. The Ledger page lists up to 17 unapproved appointments at a time:



4/4/2011	2:45 - 5:00	Jane S	Wally Smith	10.00			Emily	☐
4/5/2011	2:45 - 5:00	Henry B	○ Manny Kidds	10.00	○ Cly Entt	10.00	Emily	☐
4/6/2011	2:45 - 5:00	Henry B	○ Manny Kidds	10.00	○ Cly Entt	10.00	Kelsey	☐
4/7/2011	7:00 - 7:45	Larry W		0.00			Linda	
4/7/2011	2:45 - 5:00	Junee J	Barnaby Jones	4.00	Harriett Jones	6.00	Kelsey	☐
4/11/2011	7:00 - 7:45	Sam A		0.00			-------	

Below is an enlarged view of the appointment list for the current set of unapproved appointments. For each appointment, the date, actual start and end times, and child name are listed. The family characteristics—which determine how clients are charged—are displayed in the outlined section of each line. Finally, each line contains the name of the sitter on duty for each appointment, as well as a checkbox for approving the appointment.


Sitter's Ledger
Calendar | Ledger | Billing | Payroll | Taxes | Clientele | Setup

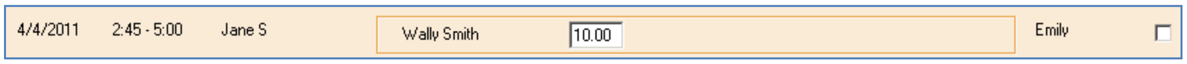
4/4/2011	2:45 - 5:00	Jane S	Wally Smith	10.00			Emily	☐
4/5/2011	2:45 - 5:00	Henry B	○ Manny Kidds	10.00	○ Cly Entt	10.00	Emily	☐
4/6/2011	2:45 - 5:00	Henry B	○ Manny Kidds	10.00	○ Cly Entt	10.00	Kelsey	☐
4/7/2011	7:00 - 7:45	Larry W		0.00			Linda	
4/7/2011	2:45 - 5:00	Junee J	Barnaby Jones	4.00	Harriett Jones	6.00	Kelsey	☐
4/11/2011	7:00 - 8:00	Sam A	Wally Smith	6.25			-------	

Incomplete ledger rows

There are two rows in the preceding example that do not have checkboxes. A checkbox will be omitted from a row if there is not enough information for which to generate a charge transaction. In the first row without a checkbox in the example, the child has not been assigned to a family. This is evident because there are no clients shown in the outlined area of the row. If Clientele information is set up completely, this situation should rarely occur. More likely is the condition in the second row without a checkbox in the example. Here, there is no sitter name, because a sitter had not been selected for this appointment type on this date. Simply returning to the Calendar page and selecting a sitter will complete this ledger row.

Family types

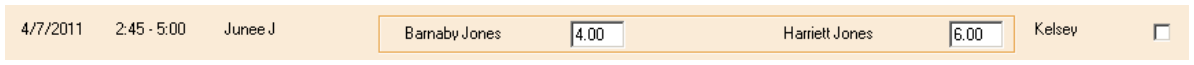
An examination of the ledger rows in the example reveals three different types of family situations. The first, represented by the first row in the example, is a family for which only one client is responsible for payment:



4/4/2011	2:45 - 5:00	Jane S	Wally Smith	10.00	Emily	☐

Although the default appointment amount will normally be what is charged to the customer, that amount can be modified here if necessary. Simply key into the box next to the client name the amount you will be charging for the appointment.

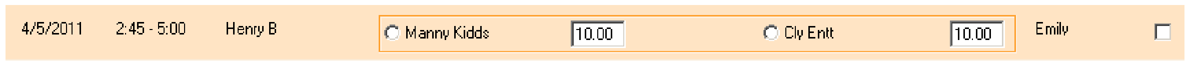
The next family type, shown below, is one in which two clients are responsible for payment, their individual amounts a percentage of the total:



4/7/2011	2:45 - 5:00	Junee J	Barnaby Jones	4.00	Harriett Jones	6.00	Kelsey	☐

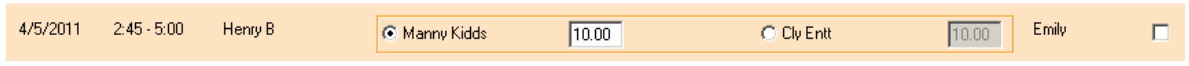
In this case, the total charge of ten dollars has been split according to the percentage (40% for the first client). Even though Sitter's Ledger has automatically calculated these charges for you, the individual amounts can be altered if necessary.

The final type of family situation is one in which two clients are responsible for payment, but they pay for individual days based on their own arrangement. A ledger row for this situation is shown below:



4/5/2011	2:45 - 5:00	Henry B	○ Manny Kidds	10.00	○ Cly Entt	10.00	Emily	☐

Here, you must select the client who is responsible for paying for this appointment. When you do so, the charge field for the other client on the row will be disabled (grayed out):

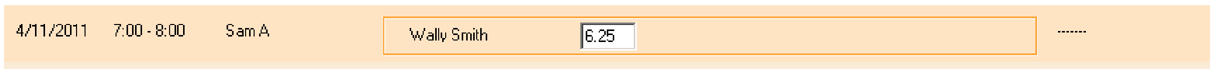


4/5/2011	2:45 - 5:00	Henry B	● Manny Kidds	10.00	○ Cly Entt	10.00	Emily	☐

(If you inadvertently click on the wrong client, just click on the other client name, and the first client's charge field will be disabled.) Here again, you can alter the amount to be charged if necessary.

Altered appointment time

The last ledger row in the example is for an appointment whose end time was adjusted, and the charge displayed reflects that change:

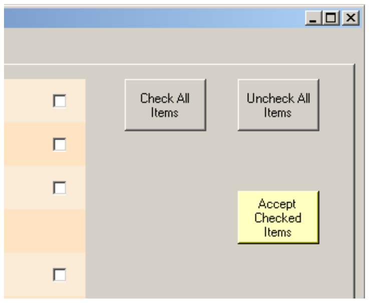


4/11/2011 7:00 - 8:00 Sam A Wally Smith 6.25 -------

In this case, the appointment was extended by one increment (15 minutes, as specified on the Setup page). The additional charge for each increment is specified in the Increment Charge field on the Setup page; in this case, the additional charge for each increment is $1.25, so Sitter's Ledger has automatically added that amount to the original base rate of $5.00.

Processing approved appointments

Now that you have reviewed, and perhaps altered, the entries in the list, you must check the checkbox for each that is correct. If all the rows with checkboxes are correct, you can click the Check All Items button at the upper right of the page (shown below) instead of checking each one individually. If you change your mind, you can uncheck them all by clicking the Uncheck All Items button.



After the checkboxes for the approved items have been checked, the final step in the process is to click the Accept Checked Items button. This will generate charges for the clients and pay records for the sitter.

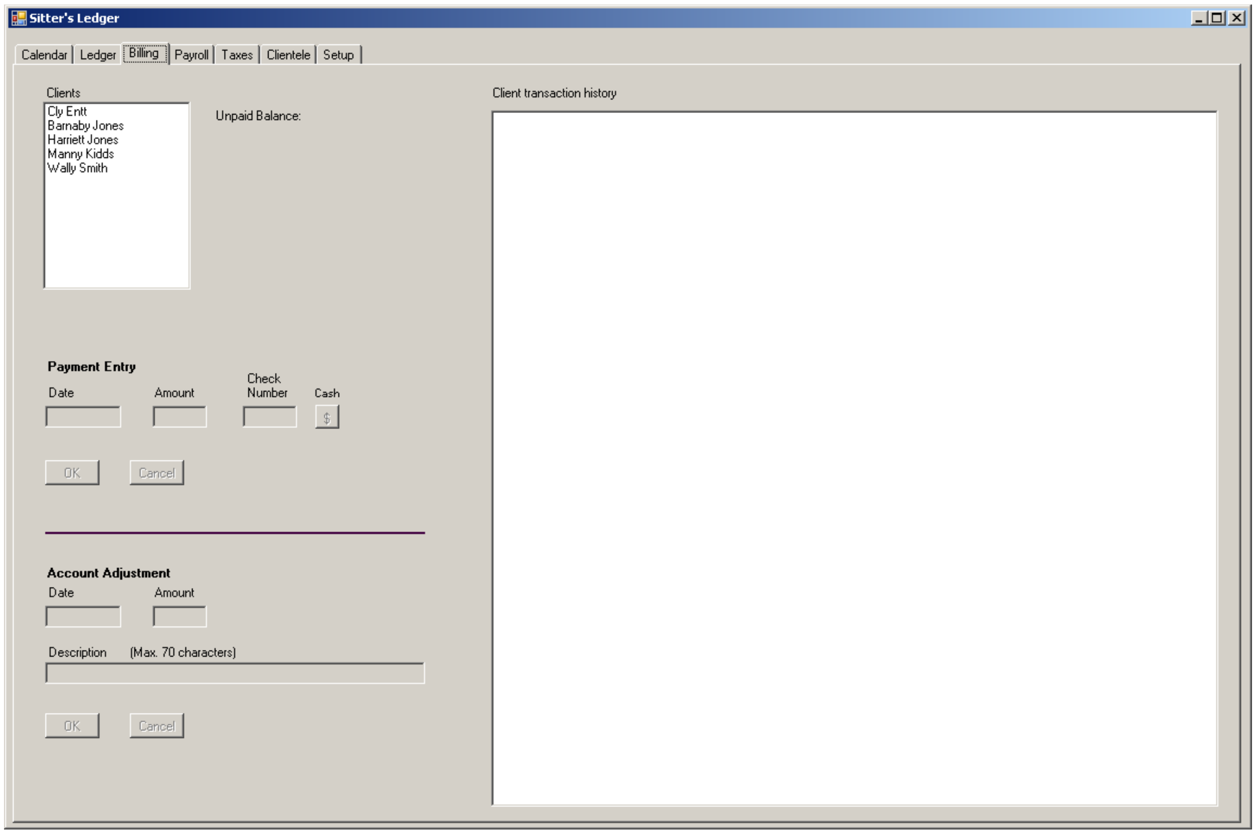
Caution: Once you have clicked the Accept Checked Items button, the checked entries cannot be undone.[1] Be sure you have reviewed and modified as needed all rows that have been checked before you use the Accept Checked Items button.

If all 17 rows on the ledger had data, you approve some or all of them, and there are more appointments yet to be approved, additional appointments will be displayed on the ledger after you have accepted those on the current ledger.

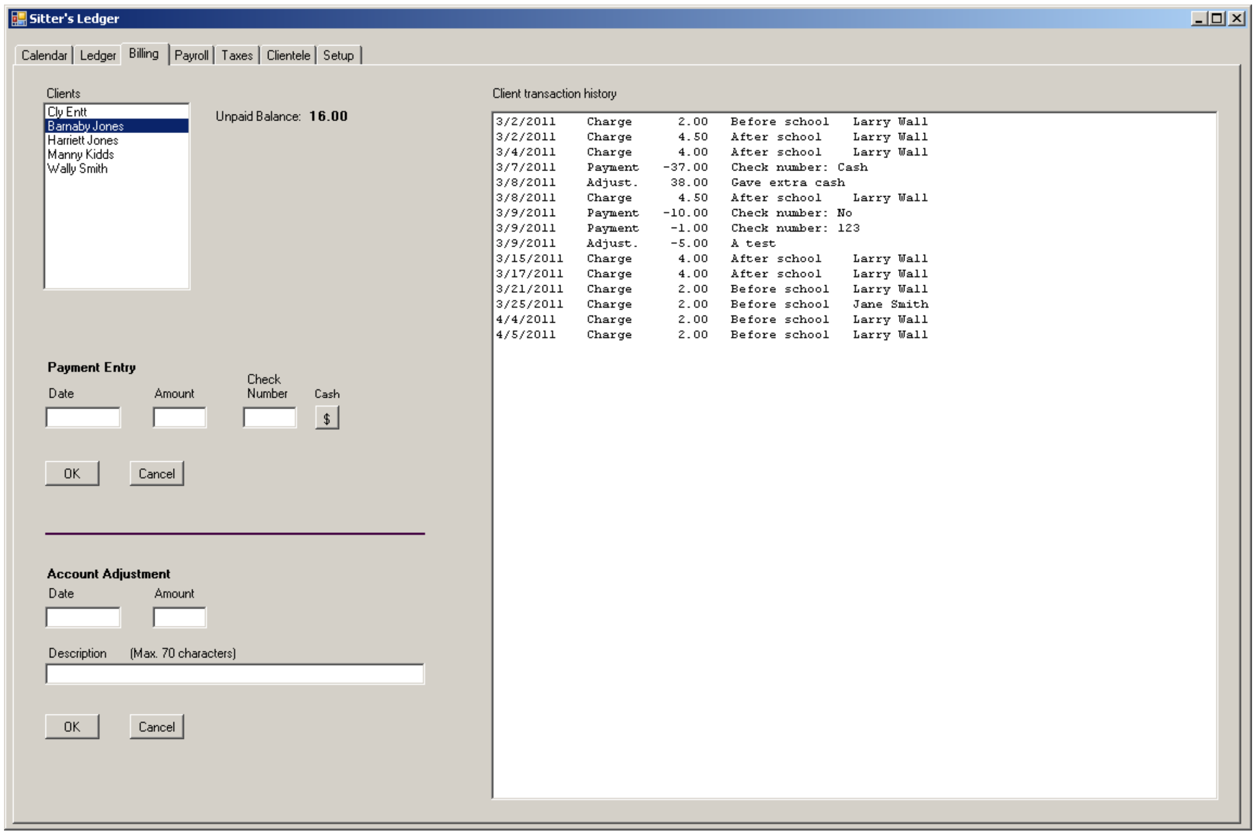
[1] The Billing and Payroll pages allow you to enter adjustments to clients' and sitters' balances, so it is possible to correct inadvertent approvals through the adjustment process.

The BILLING Page

Open the Billing page by clicking on the Billing tab. The Billing page is where you can view a client's balance due, and enter client payments and adjustments:



To view a client's information, click on the client's name in the list at the upper left of the page, as shown in the next example.



Clients: Cly Entt, Barnaby Jones, Harriett Jones, Manny Kidds, Wally Smith

Unpaid Balance: **16.00**

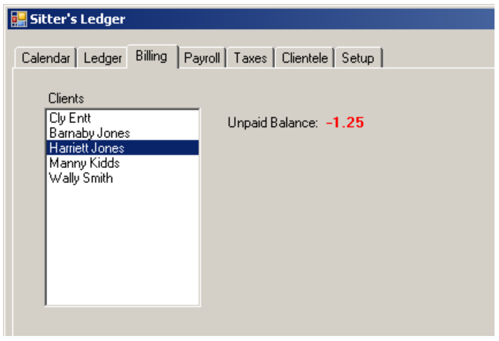
Client transaction history

```
3/2/2011    Charge      2.00   Before school   Larry Wall
3/2/2011    Charge      4.50   After school    Larry Wall
3/4/2011    Charge      4.00   After school    Larry Wall
3/7/2011    Payment   -37.00   Check number: Cash
3/8/2011    Adjust.    38.00   Gave extra cash
3/8/2011    Charge      4.50   After school    Larry Wall
3/9/2011    Payment   -10.00   Check number: No
3/9/2011    Payment    -1.00   Check number: 123
3/9/2011    Adjust.    -5.00   A test
3/15/2011   Charge      4.00   After school    Larry Wall
3/17/2011   Charge      4.00   After school    Larry Wall
3/21/2011   Charge      2.00   Before school   Larry Wall
3/25/2011   Charge      2.00   Before school   Jane Smith
4/4/2011    Charge      2.00   Before school   Larry Wall
4/5/2011    Charge      2.00   Before school   Larry Wall
```

The client's unpaid balance (amount owed) is shown, and at right is a list of transactions that have been recorded for the client. (If there are more transactions on file than fit in the translation list box, the most recent transactions will be displayed.) In this example, you can see a variety of charges (created from approved appointments), payments that the client has made, and some adjustments to the client's balance.

Each entry in the transaction history list includes the date, type, and amount of the transaction. Charge transactions include the appointment type and the child for which the charge was created. Payment entries list the check number (if any). Adjustment transactions include the adjustment description.

If a client should have a credit balance (in other words, has paid more than owed), the unpaid balance figure will be displayed in red, as shown in the image below:

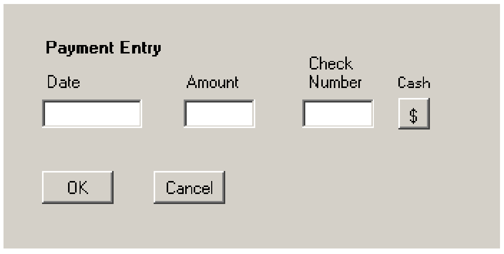


Clients: Cly Entt, Barnaby Jones, Harriett Jones, Manny Kidds, Wally Smith

Unpaid Balance: **-1.25**

Payment entry

When a client has made a payment, it is recorded in the Payment Entry section of this page, shown enlarged in the image that follows:

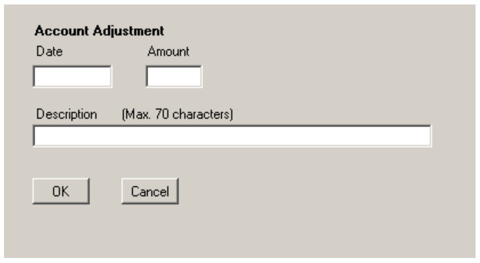


Key in the date of the payment, the amount that the customer paid, and (optionally) the check number. If the customer paid in cash, you can key the word **cash** into the check number box, or simply click the **$** (Cash) button instead, which will insert the word **Cash** into the check number box for you.

When done, click the OK button to accept the payment, which will be added to the client's transaction list, and will update the client's unpaid balance. Click the Cancel button to abandon payment entry.

Adjustment entry

There may be times when an incorrect charge or payment has been recorded. To adjust a client's balance to correct such discrepancies, click on the client's name in the list, then enter an adjustment in the Account Adjustment section of the page:

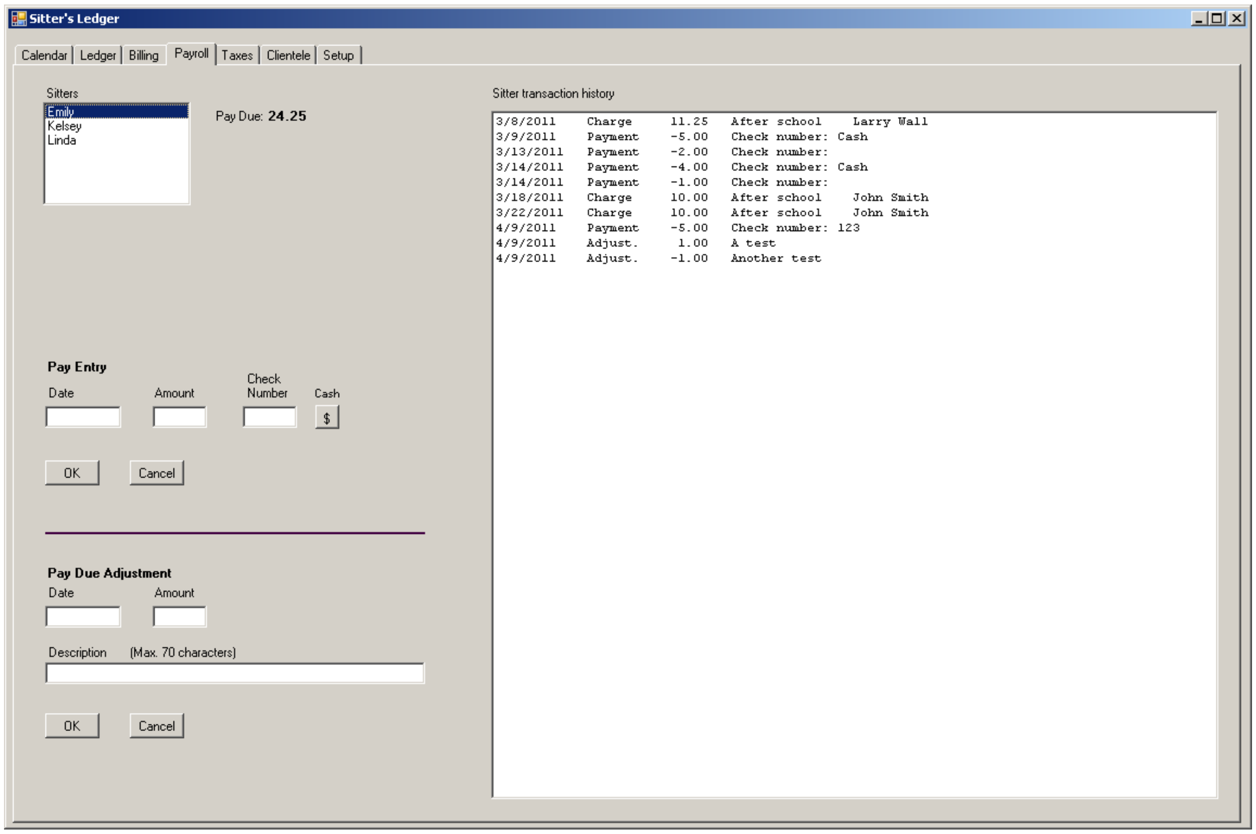


Key in the date of the adjustment, its amount, and a description of the purpose of the adjustment. A positive adjustment amount will increase the amount the client owes; a negative amount will decrease the client's balance.

When done, click the OK button to accept the adjustment, which will be added to the client's transaction list, and will update the client's unpaid balance. Click the Cancel button to abandon adjustment entry.

The PAYROLL Page

Open the Payroll page by clicking on the Payroll tab. The Payroll page is where you can view the amount due each sitter, and enter payments that have been made to each sitter. Adjustments to the sitter amount due can also be entered on this page, an example of which is shown below, with a sitter already selected:

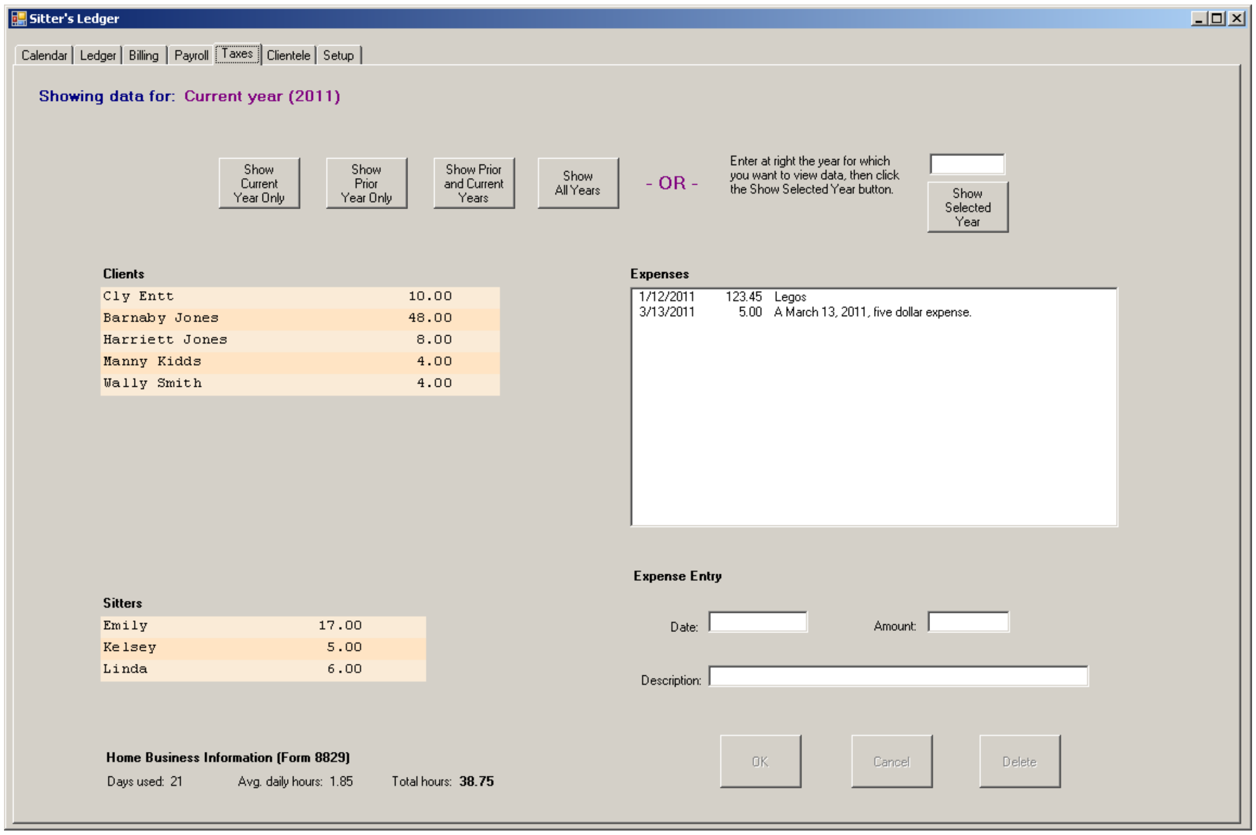


```
3/8/2011    Charge     11.25   After school    Larry Wall
3/9/2011    Payment    -5.00   Check number: Cash
3/13/2011   Payment    -2.00   Check number:
3/14/2011   Payment    -4.00   Check number: Cash
3/14/2011   Payment    -1.00   Check number:
3/18/2011   Charge     10.00   After school    John Smith
3/22/2011   Charge     10.00   After school    John Smith
4/9/2011    Payment    -5.00   Check number: 123
4/9/2011    Adjust.     1.00   A test
4/9/2011    Adjust.    -1.00   Another test
```

Similar to the Billing page, the Payroll page shows the amount the sitter is owed, the transactions recorded for the sitter, and sections for entering payments and adjustments to the sitter's pay due. Since the functions of the Payroll page mirror those of the Billing page, refer to the Billing page section of this manual for a detailed description of the Payroll page features. Note that positive adjustment amounts entered on the Payroll page increase the amount that is due the sitter; negative adjustment amounts decrease the amount due.

The TAXES Page

Open the Taxes page by clicking on the Taxes tab. Information needed for income tax returns for clients and sitters is displayed on this page:


Sitter's Ledger

Calendar | Ledger | Billing | Payroll | Taxes | Clientele | Setup

Showing data for: Current year (2011)

Show Current Year Only
Show Prior Year Only
Show Prior and Current Years
Show All Years
- OR -

Enter at right the year for which you want to view data, then click the Show Selected Year button.

Clients

Cly Entt	10.00
Barnaby Jones	48.00
Harriett Jones	8.00
Manny Kidds	4.00
Wally Smith	4.00

Expenses

```
1/12/2011     123.45   Legos
3/13/2011       5.00   A March 13, 2011, five dollar expense.
```

Sitters

Emily	17.00
Kelsey	5.00
Linda	6.00

Expense Entry

Date: Amount:

Description:

Home Business Information (Form 8829)

Days used: 21 Avg. daily hours: 1.85 Total hours: 38.75

When first displayed, the Taxes page show information for the current calendar year. On the left side of the screen, clients and sitters are listed, with the amount they paid (in the case of clients) or that they were paid (sitters). At the bottom of the left side are some statistics about how much time was spent doing child care during the year, the key item being the total number of hours. That total can be used for computing the percentage of the year during which day care was provided in the home, which is needed when preparing the income tax form for business use of a home.

On the right side of the screen is a list of child care business expenses for the year, as well as a section for entering new expenses.

Viewing data for other time periods

The year for which tax information is displayed can be changed by using the series of buttons and the year box at the top of the Taxes page. Use one of the buttons to tailor the display to the year or years for which you want to display data, or enter a specific year in the box at the right, then click the Show Selected Year button to show data for that year.

Expense entry

To add a new expense, enter in the *Expense Entry* section the date, amount, and description of the expense. The page does not have to be displaying tax data for the year for which you are entering an expense. However, you will see the newly added expense in the Expenses list only when the year for which the expense was entered is being shown.

When you are done entering the expense, click the OK button. Use the Cancel button to abandon the entry.

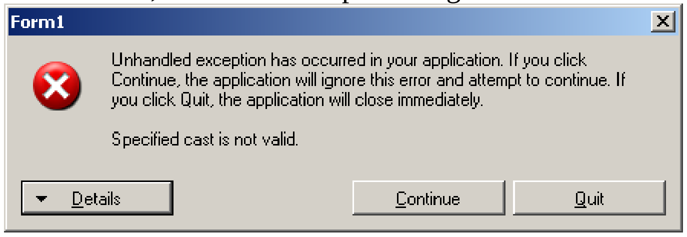
Change an expense
To change the details of an existing expense entry, click on its line in the Expenses list, then change the information in the Expense Entry fields. When you are done changing the expense information, click the OK button. Use the Cancel button to abandon the changes.

Delete an expense
To delete an expense, click on its entry in the Expenses list, then click the Delete button.

Larry Overkamp
CSC 450 Spring, 2011

Project Journal

Date	Time Spent (hours)	Activity
January 22	1.5	Reference book research to re-familiarize myself with Windows programming and database access concepts.
January 23	1	Reference book research.
January 24	.5	Reference book research.
January 25	1 1 1	Reference book research. Database design research. Tried out in Visual Studio some Windows form controls with which I was not familiar.
January 26	2	Tried out some more new Windows form controls.
January 27	2	More Windows form control experimentation.
January 28	5	More Windows form control experimentation.
Had to relearn database access coding. Lots of trial and error to get a connection established and to retrieve data. Couldn't successfully compare dates in SQL queries. Much Internet research.		
January 29	6	More Windows form control experimentation.
Learned about the Panel control, along with its properties, which could be used as the basis of the calendar. Learned how to make collections of form controls from http://msdn.microsoft.com/en-us/library/aa289500%28v=vs.71%29.aspx. This could allow repeating controls to be referenced by index, which would allow the use of code common to each iteration. 1/29. Couldn't get updates to an experimental database to work. Found that the database in the bin dir was getting updated, but every time I tested, my new records were gone. This is because Visual Studio was pulling in the database from the Project directory structure and not from the bin folder. Found that concatenation in an SQL SELECT statement won't work if one of the fields returned is null; the concatenated field is then null.		

Date	Time Spent (hours)	Activity
January 30	6	More Windows form control experimentation.
Attempting to copy a null database field to a string object failed with a casting error. Tried comparing to null, but that still didn't work. Learned that DBNull.Value had to be used instead of null when checking to see if a database field is null. Had to find out how to prevent firing the SelectedIndexChanged event of listboxes when they're bound to a datasource, because it was producing undesirable results:  Form1 Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately. Specified cast is not valid.		
January 31	.5 1	Updated the initial project proposal. Initial in-class presentation preparation.
February 1	2	More Windows form control experimentation.
Finally figured out how to do SQL SELECT date comparisons. Date values for Access database SQL queries have to be wrapped in # signs. For example, #02/01/2011#.		
February 2	2	More Windows form control experimentation.
Learned about the ListView control, and experimented with options.		
February 3	2	More Windows form control experimentation.
Learned about the MaskedTextBox control. Was going to use for the expense date text box, but decided against it to allow for my own date validation logic and, more importantly, to allow months and dates to be entered without leading zeros. Learned that when selecting an item in a listview control after another item was previously selected, the SelectedIndexChanged event fires twice: once when the previous item is de-selected and once for the newly-selected item.		
February 4	2	Coding.
February 5	4	Coding.
Learned how to draw a line on a form at design time, courtesy of http://blog.jtbworld.com/2006/01/draw-line-at-design-time-in-visual.html: In Visual Basic 6.0 there was a **Line** control that could be used to draw a line on a form. In VBA, VB.NET, Visual Basic 2005 and other similar programs there is no line control that you can use. The most lightweight and easy thing to do is to use a **Label** control. Set Height or Width to 1 depending on if you want the line horizontal or vertical Clear the Caption / Text Set the BorderStyle to None Set AutoSize to False Set the BackColor to the color of the line		
February 6	5	Coding.
I put a groupbox on the main form just to indicate where the panels will be. It's Visible property is set to False, though.		

Date	Time Spent (hours)	Activity
Learned about the PadLeft method for strings, because I wanted calendar day numbers to be right-justified. Had to set “Copy to Output Directory” property to “Copy if newer” in order to have data entered in during testing remain. The property was “Copy always”.		
February 7	2	Coding.
February 8	1	Coding.
February 9	2	Coding.
Couldn’t use ConfigurationManager class because it wasn’t found, even with “using System.configuration;”. Found online that I had to add a reference to the project: System.configuration.dll. Also had to use “using System.Collections.Specialized;” to use the NameValueCollection object to loop through the appSettings collection. [Don’t know if I’ll end up using this. Added this because I wanted to see if I could read and write Setup information like default rates, etc.] To prevent the application from generating the message "File has been modified outside of source editor. Do you want to reload it?" when the config file was updated via the program, did Tools menu, Options… item, Environment section, Document sub-section, and checked “Auto-load changes, if saved”.		
February 10	2	Coding.
February 11	5	Coding.
Found out that you can’t remove a ListView control if it has an item selected. Had to call its Clear() method before removing it.		
February 12	5	Coding.
Found the Microsoft ImageLibrary, because I wanted to see if there were some images I could use on some buttons.		
February 13	5	Coding.
February 14	2	Coding.
February 15	3	Coding.
Discovered that “session” was a reserved word in Microsoft Access when trying to INSERT with a field named “session”, and I got a syntax error. Changed the name of the field.		
February 16	3	Coding.
February 17	3	Coding.
February 18	1	Coding.
February 19	5	Coding.
February 20	5	Coding.
February 21	1	Coding.
February 22	2	Coding.
February 23	2	Coding.

Date	Time Spent (hours)	Activity
February 27	6	Coding.
February 28	1.5 2	Coding. Changes to project spec document.
March 1	2 2	Coding. Updates to project spec document, and work on in-class presentation.
Found out how to use regular expressions for validating input.		
March 2	1	Coding.
March 3	2	Coding.
Found out about the class initializer mechanism, and used it for the Transaction class.		
March 4	1.5	Coding.
March 5	8	Completion of modifications to the project spec document.
March 6	6	Coding.
March 7	3	Coding.
March 8	2	Coding.
March 9	3	Coding.
Found out how to get the identity number of the just-inserted database record.		
March 10	1	Coding.
3/10 Found out how to convert integers to enum values: http://www.switchonthecode.com/tutorials/csharp-snippet-tutorial-how-to-get-an-enum-from-a-number and enum entries to integers: http://msdn.microsoft.com/en-us/library/sbbt4032.aspx		
March 11	2	Coding.
March 12	4	Coding.
Learned how to create a listbox with a different background color for each row: http://msdn.microsoft.com/en-us/library/system.windows.forms.drawitemeventargs.drawbackground.aspx http://www.csharphelp.com/2006/09/listbox-control-in-c/ and to keep the list item from being highlighted with a blue background when selected (which happened even if the listbox's selectionmode was set to None), changed the state value in the new DrawItemEventArgs call, the solution to which was found here: http://www.vbdotnetforums.com/graphics-gdi/44467-listbox-selectionmode-none-question.html		
March 13	5	Coding.
March 14	4	Coding.
March 15	3	Coding.
March 16	4	Coding.
March 17	3	Coding.

Date	Time Spent (hours)	Activity
March 18	2	Coding.
March 19	5	Coding.
March 20	5	Coding.
March 21	1.5	Coding.
March 22	3	Coding.
March 23	3	Coding.
March 24	4	Coding.
March 25	1.5	Coding.
March 26	3.5	Coding.
March 27	3	Coding.
March 28	2	Coding.
March 29	2	Coding.
Learned how to display a dialog box while another process was running. To be used to ask the user to wait while data is being loaded in certain circumstances. http://social.msdn.microsoft.com/Forums/en-US/winformsdatacontrols/thread/7c41ca82-34f7-4484-8ff3-7ed6087ff7b5/ [Ended up not using this functionality because the program thought the database connection was still open. Couldn't get it to work.]		
March 30	3	Coding.
March 31	2.5	Coding.
Learned about BindingList and used it for childrenInFamily and childrenNoFamily.		
April 1	1.5	Coding.
April 2	3.5	Coding.
April 4	2 1	Coding. Updates to project spec document based on instructor feedback to previous version.
April 5	.5 .5 .5	More updates to project spec document. Preparation for second in-class progress demonstration. Work on final presentation.
April 6	.5 .5 2	Attempted to create an installer. Integrated testing. Attended second in-class progress demonstration session.
Used these web pages as a reference when trying to build the installer. http://www.dreamincode.net/forums/topic/58021-deploying-a-c%23-application-visual-studio-setup-project/ http://msdn.microsoft.com/en-us/library/s2esdf4x(v=vs.80).aspx		
April 7	.5 .5	More work on trying to build an installer. Coding.

Date	Time Spent (hours)	Activity
	2 1 .5	Integrated testing. Binder preparation. Work on final presentation.
April 8	1	Binder preparation.
April 9	11	Integrated testing.
April 10	3	Integrated testing.
April 11	1 2	Integrated testing. Work on User Manual.
April 12	3	Work on User Manual.
April 13	.5 2.5	Work on User Manual. Binder preparation.
April 14	3	Binder preparation.
April 15	1	Binder preparation.
April 16	1.5 7	Binder preparation. Work on final presentation.
April 17	2	Work on final presentation.
April 18	1 2	Binder preparation. Work on final presentation.

Larry Overkamp
Capstone Project
CSC 450   Spring, 2011

Reflection

This course was easily the most demanding I have taken during my time at Carroll. Being responsible for project design, project management, coding, testing, and documentation—tasks usually spread out among a team of individuals in the real world—was a test of resolve and endurance. My personal situation (full-time job, family responsibilities) compounded that challenge. To be honest, there were times during the semester—especially during those occasions when a seemingly irresolvable problem arose—that I felt the project would come up short of expectations. In the end, though, the goals outlined in the project proposal were met.

Since it had been some time since I had done the type of coding needed for the project I selected, I realized I would have to brush up on my Windows forms development, C# language, and database access knowledge. I also knew that the Visual Studio environment and .NET features had been enhanced in the recent past, so I sought to familiarize myself with those features—especially with new and enhanced form controls—so I could use the most appropriate elements in my coding. To that end, I planned to spend two to three weeks perusing a variety of sources—from books to the Web—to firm up my base of knowledge.

Application design was reasonably straightforward. The intended user was readily available for consultation. On top of that I had seen the business in operation to some extent, so I was able to get a good sense of the various processes that were involved. From there I could outline the entities I thought I'd need, and design the database. Although my original design was close to the finished product, there were a couple times during development that I realized some significant modification to my database design would be necessary. For example, I had originally included a couple junction tables in my design that, based on the eventual relationships involved, were unnecessary and were therefore dropped. There were also some changes after follow-up discussions, such as eliminating the need for client addresses and adding the ability to record notes for individual children and clients.

The primary objective of the user interface design was to make it as simple and as intuitive as possible. The intended user of the application does not have extensive experience with computer applications, so I strove to create an uncomplicated interface with few optional features. Segregating the functions into different tab pages, which are presented in a sequence through which business activities normally occur, helped achieve that goal, as did the inclusion of data entry shortcuts.

The application development proved to be the most challenging part by far. In the interval between my last Visual Studio activity and the present semester, my skills in that area deteriorated somewhat. That was overcome by much research and some trial and error, but there were several times that I had to wrestle with issues and with trying to find a way to do in Visual Studio what I was envisioning. Below are some examples of those challenges:

Iterative controls

As I started to design the application's Calendar page, I wanted to be able to treat each calendar cell as one of a group, as in an array, so I could use iterative code to deal with the cells and their contents without having to create individually-named controls for each cell. This was a concept I was not familiar with, and I didn't know if it was even possible. Fortunately I found some information on a few Web sites that explained how to accomplish this objective. Using this method I am able to access the controls using an array index. The same concept was used for the rows on the Ledger page. As a side note, the parent control for both the Calendar array and the Leger array is the Panel control, with which I was unfamiliar. It turned out to be the ideal container for my needs.

SQL date queries

Several queries in the application need to return data for a given range of dates. My initial query syntax failed, so I turned to the Web. Some solutions mentioned special date formatting or particular SQL keywords, but none of suggestions worked. Ultimately I discovered the correct syntax needed when doing SQL date comparisons in queries to Access databases: wrapping the date in a pair of # characters.

Null database fields

I was getting casting errors when trying to initialize a string variable with a database field that held a null value. Comparing that value to null did not work, so I had to do some research. I discovered that the comparison had to be done with the DBNull.Value property.

ListView behaviors

I ran into a couple behaviors when using the ListView control. The event handler for the SelectedIndexChanged event was firing more often than I was clicking on items. I eventually discovered that the event fires also for when the previously-selected item is deselected, which happens when a new item is selected. I was able to program around this to achieve the desired actions. The second behavior was encountered when I tried to remove a ListView control from the form, and doing so would generate an exception. For this issue, I discovered that the ListView items have to be cleared before the control can be removed.

A reserved word

One of the perplexing puzzles I encountered was the triggering of a syntax error on a database INSERT. I displayed the complete SQL statement, and my examination revealed no obvious mistakes. I played around with it a bit, but couldn't get past the syntax error, so I had to turn to the Web. The problem turned out to be that I was using the word "session" as one of my database fields—a word that is a reserved term in Microsoft Access. Changing the column name resolved the issue.

Part of good programming practice is to maintain a consistent way of doing things throughout the application. Knowing that a certain type of logic is done the same way regardless of where it is makes it easier to maintain the code. This is even more important in an object-oriented environment, where one must be confident that each object has been completely defined within its own class. While my application runs as expected, it is evident through a review of my code that I have fallen short of these objectives in several places.

The development process itself was on the whole effective. Application requirements were well-defined, as very little adjustment to the initial set of requirements. Unit testing of functions was done as they were developed. After all coding was done, and the application deemed complete, all the scenarios outlined in the separate integrated testing document were performed. While some issues arose (proving that the testing process was both needed and effective), there were relatively few given the number of scenarios run.

For the most part, all functionality outlined at the beginning of the project has been included in the final product, and it is a usable application. There were a couple items not delineated in the project specification document that I intended to include, but could not due to lack of time. Probably the most significant is the ability to purge old data. While the amount of data recorded during a typical year is relatively small, it is not hard to imagine that there might be degradation in performance after two or three years. The other omitted item was a method for tracking time spent using the application which could be used to calculate a deduction for income tax purposes for the depreciation on the home PC used for the child care business. Because the time spent in the application will be negligible, such a deduction would likely be small, so this omission is not significant.

Other ways to enhance the application emerged over the course of development, but time or malfunction conspired against their inclusion:

Installation Routine

The present method of installing the application is to copy the files on the program disc to a folder on a PC, and to optionally create a shortcut to the executable. An attempt was made to create a Visual Studio Setup project, but it wasn't completely successful, so the effort was abandoned.

Calendar Display Efficiency

As it is, when the user changes the calendar by going back or forward by a week, the rebuild of the calendar is done by re-accessing the database for each date. Since navigation is by a week at a time, two weeks from the previous display remain. The data from these could be retained in memory, reducing the database access and thus speeding up the redisplay.

Inactivation of primary entities

The deletion of key items like clients and children is provided. However, such entities cannot be deleted if dependent database records exist. Should a given family, for example, discontinue using the child care services sometime during the year, that family's children would still show up in the list of children for which appointments can be made. While that's not a serious situation, unnecessary data items just add clutter and can be inadvertently selected. The envisioned enhancement would allow the inactivation—while retaining their records in the database—of entities like families, client, and children, preserving historical data relationships. Inactivated items could, in turn, be excluded when building selection lists, thus eliminating the clutter.

Ledger row enhancements

While the entries on the Ledger page allow the amount charged to be altered, it does not provide a means to delete an appointment nor to change the sitter. Those operations can be done on the Calendar page, but having them available on the Ledger page also would be a useful improvement.

Meal/Snack Count

Another permitted income tax deduction for an individual who runs a home-based day care is for meals and snacks provided. While actual costs can be calculated, the IRS has a standard rate that can be used for meals and snacks. The day care provider need only know the number of meals and snacks given, and multiply that by the standard rates. A feature for recording the number of meals and snacks given could be incorporated into the application. The totals would then be presented on the Taxes page along with the other tax-related information.

Online help

Help text, tooltips, or the like that are accessible via the GUI itself would add a nice touch to the application.

"Please wait.."

The duration of some operations, particularly those that retrieve large amounts of data, as on the Billing, Payroll, and Taxes pages, will increase as the database does. Displaying a dialog box asking the user to wait during these functions would let the user know that the application is busy and not just "frozen". An attempt was made to incorporate this feature using components available in Visual Studio, but it was ultimately unsuccessful and had to be abandoned.

Capstone Evaluation Rubrics
Self Evaluation

Project Appropriateness	5
Project Planning and Follow-through	4
Software Analysis and Design	4
Code Quality	3
Software Usability	5
Project Verification	5
Effort	5
Binder	5
Presentation	4[1]

[1] Since this document was prepared before the final presentation, this score is based on the other items mentioned in the rubric document for this rubric, namely, attending and being prepared for all scheduled meetings and status presentations.