Invoice Demo

Lincoln Stoller, Ph.D

Contents

Overview

The Invoice Demo application illustrates a powerful and extendible method for handling multi-line invoice information. It also demonstrates important programming, and user interface techniques. These techniques, listed below, are discussed in the following sections.

Programming

- Related table structure
- 4D Transactions
- ID assignment
- Subtables
- Modularized code
- Distributed methods

Interface

- Control screen interface
- Tab and popup controls
- Resizable forms
- Enterable included forms
- Clairvoyant look-up
- Process interaction

The demo is built from 4th Quarter® Accounting Solution, an accounting platform written in 4D and marketed by Braided Matrix, Inc. for high-end custom applications. While the invoice example is limited in scope, it is not a toy system. It is the same invoicing system that we use at Braided Matrix to support complex and comprehensive 4D applications.

To learn more about how invoices can be integrated into a comprehensive accounting system, refer to the article "Accounting Transactions IV: Invoices" located on this volume. If you're interested in the 4th Quarter Accounting Solution application please contact me at Braided Matrix. We can be found on the web at www.4thquarter.com.

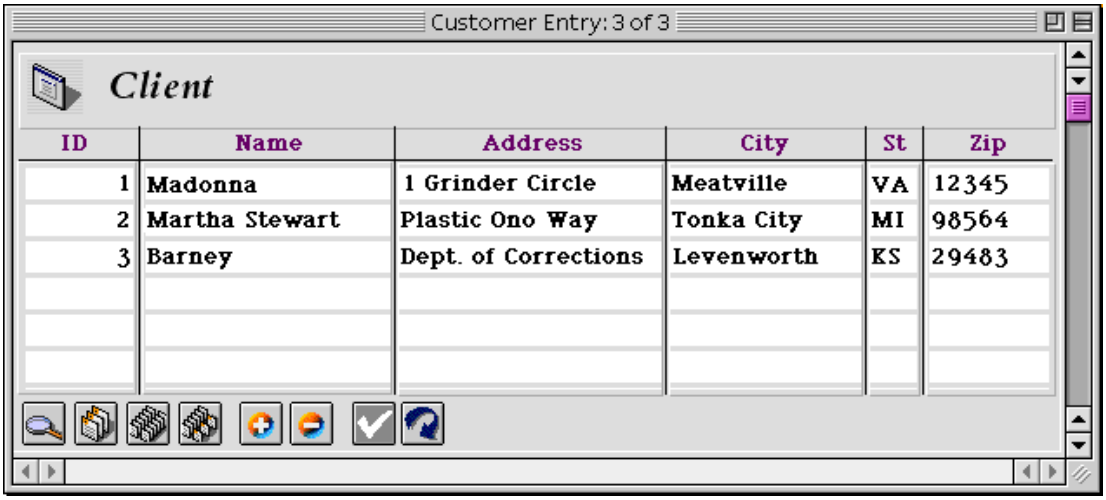
Running the Application

The demo opens in the custom menus environment, where the only means of accessing the data is through the menus and forms that the designer has provided. You navigate through the application from the control screen. The first page of the control screen provides access to the client, inventory, and invoice areas. You can also reach these areas by selecting one of the items under the Tables menu.

You reach the second page of the control screen by pressing the arrow icon in the upper left-hand corner. The second page provides some brief instructions on how to get in and out of the design environment.

Clicking on either the client, inventory, or invoice icon opens an output or list form. This output form appears in the same window as the control screen. For the most part the Invoice Demo is a single-window application.

From the output forms you can add, modify, or delete records from the three main areas. The application also uses tables internally for the management of invoice data. These are the System_Default, ID_Number, and sfLineItems tables. These tables are handled by the application, and their role is largely hidden from the user.



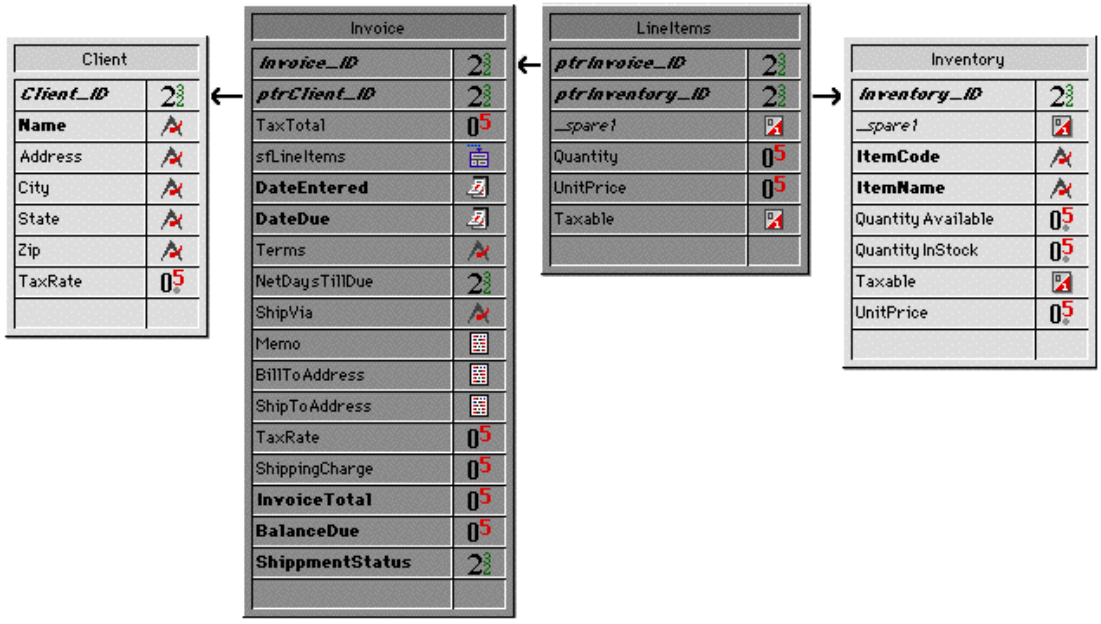
Finally, in the About dialog, I've provided a demonstration of how you can implement simple animation using two 4D processes. The About dialog is accessed through the "About the Invoice Demo..." item under the Apple menu (MacOS).

The whimsical animation in the about box offers an example of how two processes can interact. The example is easy to follow because it does not involve any data management.

Related file structure

The four central tables in the Invoice Demo are the Client, Invoice, Line Item, and Inventory tables. These tables illustrate parent-child, or one-many relations, as follows:

Parent (One)	Child (Many)
Client	Invoice
Invoice	Line Items
Inventory	Line Items

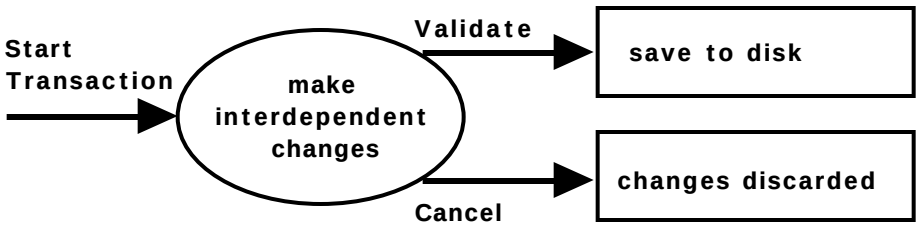


Notice that the invoice table stands in a one-to-many relationship to the line items, and a many-to-one relationship to the clients. This means that you can have many line items related to one invoice, and many invoices related to one client. The invoice-line item record structure establishes a relationship between inventory items and the clients who have placed orders for those items.

4D Transactions

4D Transactions provide a means of buffering changes to the datafile. When you start a 4D transaction, all changes that are saved are stored in memory on the local machine until the transaction is completed. The modified and saved records remain locked to all other users on the network. When the transaction is validated, all changes are written to disk at once. If the transaction is canceled, the datafile is left unchanged. In either case all the records locked during the transaction are unlocked.

4D transactions are indispensable for handling record entry in multi-user applications. However, they also play an important role in the context of a single user application. You use a 4D transaction whenever you need to save changes to records that are subject to some collective condition. In the case of the invoice, all the items ordered must be in stock, or the whole invoice cannot be entered. Using a 4D transaction allows you to check and update each line item in sequence. If a particular item is not in stock the transaction is canceled and the changes are "rolled back."

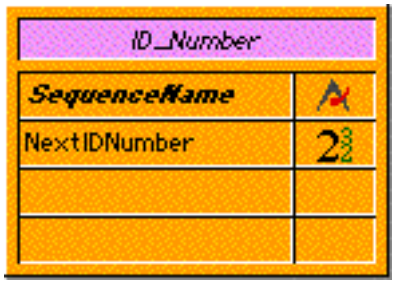


The code that manages the 4D transaction is located in the ICHandleEntry procedure of the invoice demo. For more information on 4D transactions refer to the sidebar in the "Related Entries Using Subrecords" article.

ID assignment

ID numbers are the primary key values that distinguish records in different tables. These play a central role in relational database design. 4th Dimension automatically generates a separate series of unique numbers for each file. These numbers are returned by a call to 4D's Sequence Number function. However, most developers find it advantageous to manage their own sequence numbers. This allows you to support any number of sequences, and to increment and assign them whenever and however you want.

The common means of supporting your own ID numbers is to create an ID number table. This is the ID_Number table in the invoice demo. This table contains two fields: a sequence name field that distinguishes the different ID sequences, and a NextIDNumber field that contains the next unique number in that sequence.



A certain subtlety is required for managing ID number records. This is because these records are centrally important (you cannot enter a record if it cannot be assigned an ID), and because they are a potential bottleneck for multi-user applications (every user must gain access to the same ID number record in order to secure a new ID number).

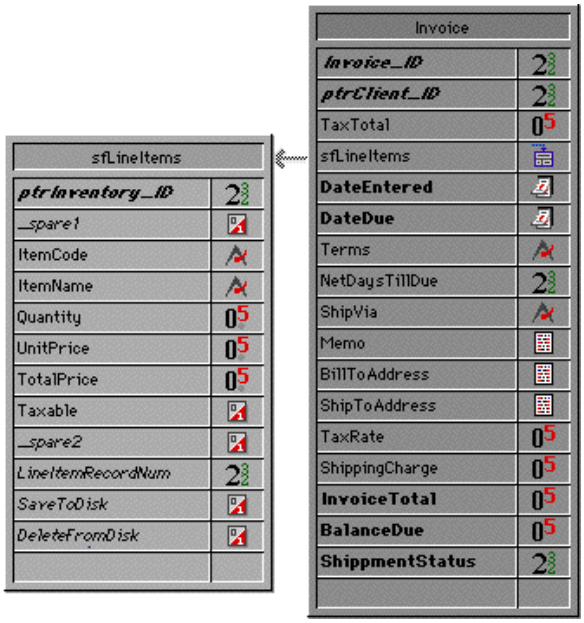
The methods you create to handle ID numbers must ensure that no two users are

ever assigned the same sequence number, and that sequence number records are kept locked as briefly as possible. Since the possibility of locked sequence number records always remains, you must also implement some means of gracefully informing the user when such a situation occurs.

All of these requirements are met by the SYIncrmntSeqNum method. For an example of how it is used, refer to the code in the CLInitializeVars method.

Subtables

The invoice table is related to a subtable called "sfLineItems". This subtable is used as a buffer for changes made to the related LineItems records. The subtable provides temporary storage for changes made to records in the LineItems table stored on disk. The subrecord entries themselves are created only for the purposes of data entry, and are not stored as part of the datafile.



This kind of buffered entry allows you to make incremental changes to the related records, then save those changes all at once when the invoice information is saved. Buffered entry is more than a good idea, it is required to preserve data integrity. To learn more about this technique, refer to the article "Related Entries Using Subrecords" located on this volume.

Using subtables in this manner is one of the best tools for entering related records. The subrecords are memory-resident, and they have a record structure like the actual tables. In addition, 4D gives us powerful tools for subrecord entry and display that are not available for arrays.

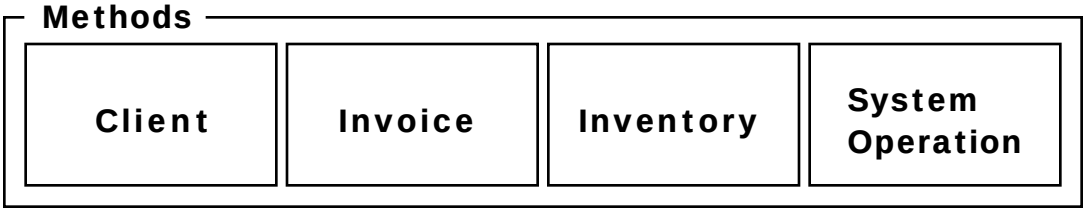
Modularized code

4D provides several built-in classes of objects from which all 4D databases are constructed. The major classes are tables, forms, and methods. In addition there are menus, lists, styles, and a large variety of interface controls.

These classes enable you to manage the pieces of your application. It is also useful

to subdivide the objects that you create within these classes. The methods used in the invoice demo span four separate areas of function, which I will refer to as "modules." These are the client, invoice, inventory, and system modules. The module to which a given method belongs is indicated by the first two letters of its name, as shown in the following table.

Module	Method prefix	Nº of methods
Client	CL	5
Invoice	IC	8
Inventory	IN	5
System operations	SY	11



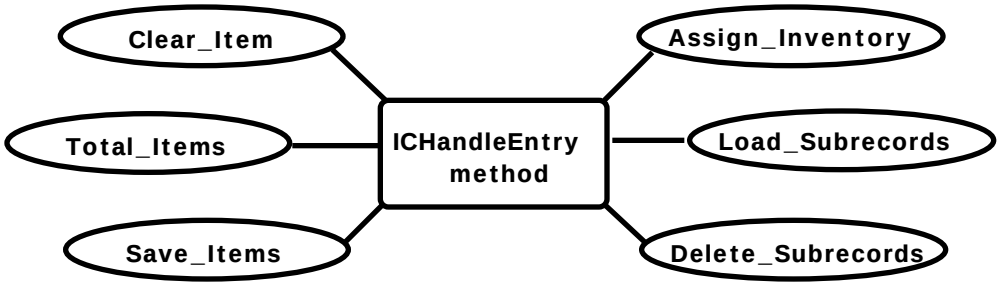
Distributed methods

In addition to these four modules, most methods perform several separate but related operations. That is, the method will perform one of a set of different operations depending on a value that's passed as its first parameter. Consider the ICHandleItems method whose "case of" structure is shown below.

```
`ICHandleEntry
Case of
:($1="Total_Items")
-- code --
:($1="Save_Items")
-- code --
:($1="Delete_Subrecords")
-- code --
:($1="Load_Subrecords")
-- code --
:($1="Assign_Inventory")
-- code --
:($1="Clear_Item")
-- code --
End case
```

This method handles the subrecords that are used to buffer changes made to records in the LineItems file. Various different operations must be performed to support this use of subrecords, as can be seen from the different action codes passed in the first parameter (the value of the first parameter is stored in the local variable named $1). While these operations are distinct, being performed at dif-

ferent times, the are also closely related.

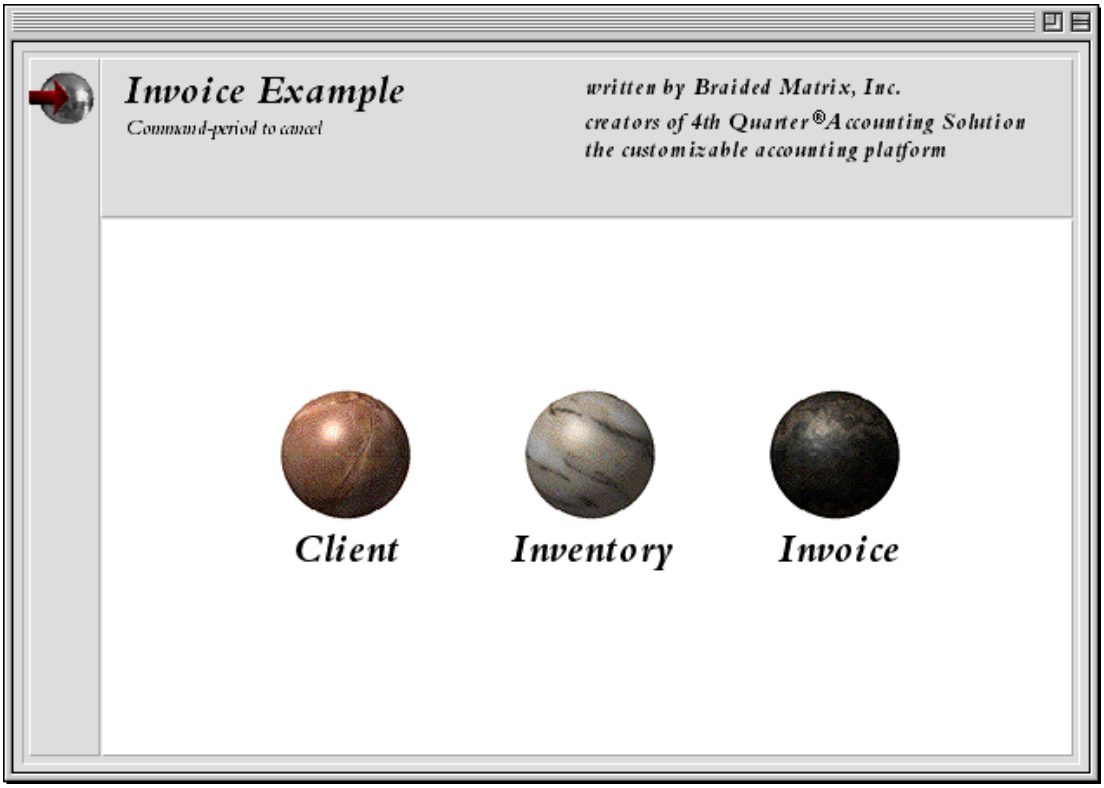


Putting related operations in a single method makes it much easier to coordinate your code. If you need to add or change a field, this method makes it easier to apply all the necessary changes. A similar reasoning leads to the use of "object methods" in object oriented programming languages. Using this method in 4D yields many of the same code management benefits.

Interface Elements

Control-screen interface

When the application is launched it performs some system record keeping, assigns global variables, then launches the control screen process. This process opens the control screen shown below.



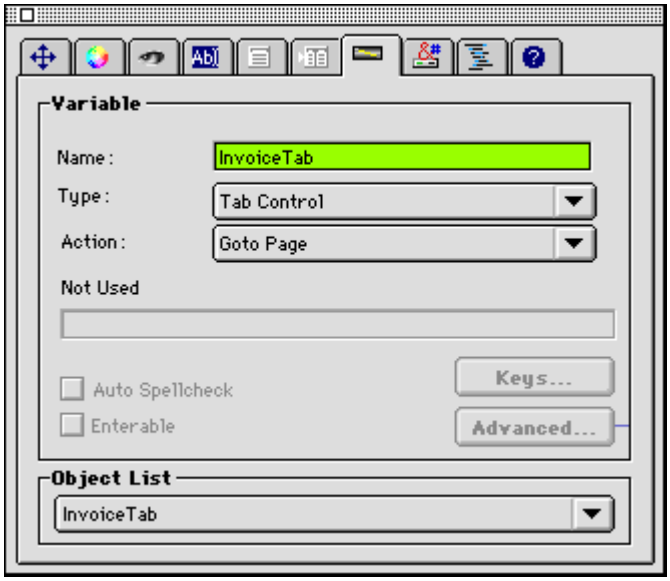
Clicking on the icons for the different areas opens a list form that replaces the control screen. The list form is actually opening in the same window used to display the control screen. You don't need to exit the control screen dialog — the list form opens in front of it. When you exit the list form you return to the control screen.

The implementation of a control screen is made simple by displaying all the data in the same window. The control screen can be modified to support multiple windows. To do this you will need to spawn a new process for each area, opening a separate window in each.

Tab and popup controls

Most of the examples of control objects appear on the invoice entry form. Here the tab control is used to navigate between the invoice's client and line item pages. The tab object is drawn on page 0 of the form. The tab's action is set as "Goto Page" in the object inspector. When set in this manner,

clicking on the tabs automatically takes the user from one page to the next.

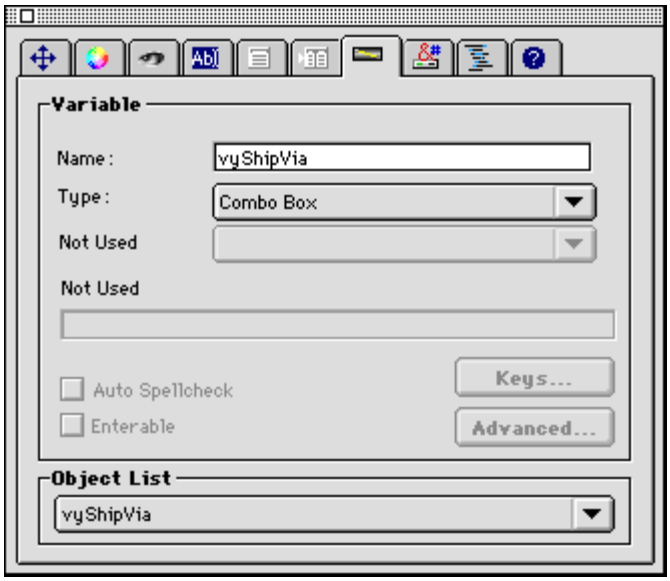


The tab labels are set in the script of the tab itself. This script has two actions, one that sets up the tab when the form is first entered, and another that clears memory when the form is unloaded.

```
Case of
  : (Form event = On Load)
    InvoiceTab:=New list
   APPEND TO LIST (InvoiceTab;"Customer";1)
   SET LIST ITEM PROPERTIES (InvoiceTab;1;True;0;0)
   APPEND TO LIST (InvoiceTab;"Items";2)
   SET LIST ITEM PROPERTIES (InvoiceTab;2;True;0;0)

  : (Form event = On Unload)
   CLEAR LIST (InvoiceTab)
End case
```

A combo box is used on page two for the entry and display of shipping information.

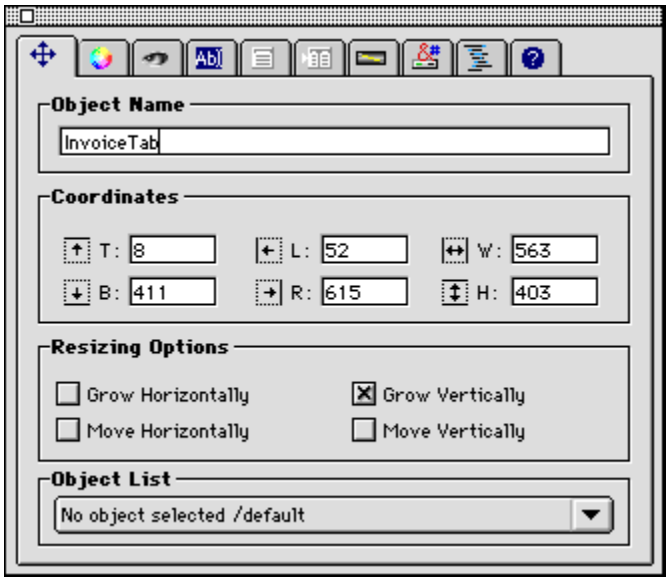


The object is associated with the vyShipVia array, and the elements assigned to this array appear in the popup menu of the combo box. Values are assigned to this array from the "ShipVia" list. This is done in the ICHandleEntry method at the time the form is loaded. It is accomplished with a single call to **LIST TO ARRAY ("ShipVia";vyShipVia).**

Resizable forms

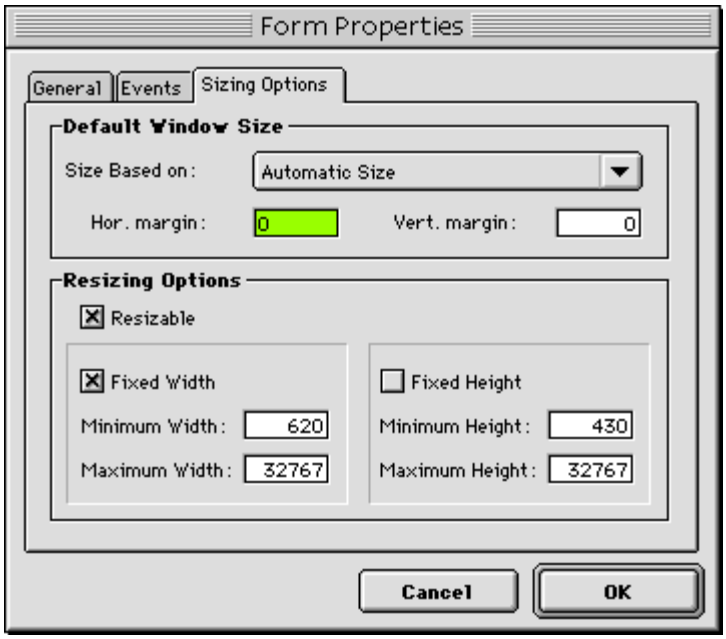
Both pages of the invoice entry screen are resizeable. The objects on these pages move or scale as the window grows. On the second page the line item subtotals move with the lower edge of the window. The included form expands with the height of the window.

Object motion properties are set on the object properties page of the inspector, as shown below. You must specify the properties of each object correctly. Otherwise objects may move out of alignment or become overlapped when the window size is changed.



If you select multiple objects (by holding the shift key down while clicking on a sequence of objects), then the properties you set in the inspector will apply to all selected objects.

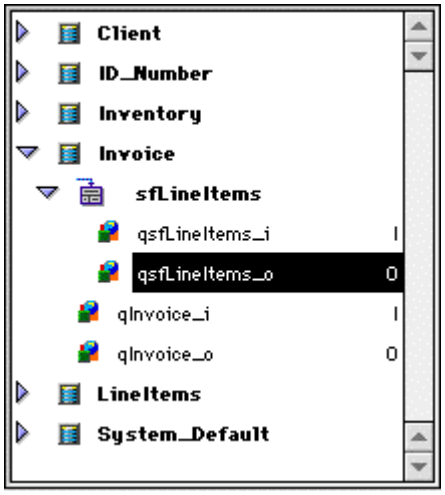
The scaling properties of the form itself must also be set. This is done from the Sizing Options page of the Form Properties window. These settings apply to all pages of the form. The resizing properties of the form and the objects on the form must be set for resizing to work.



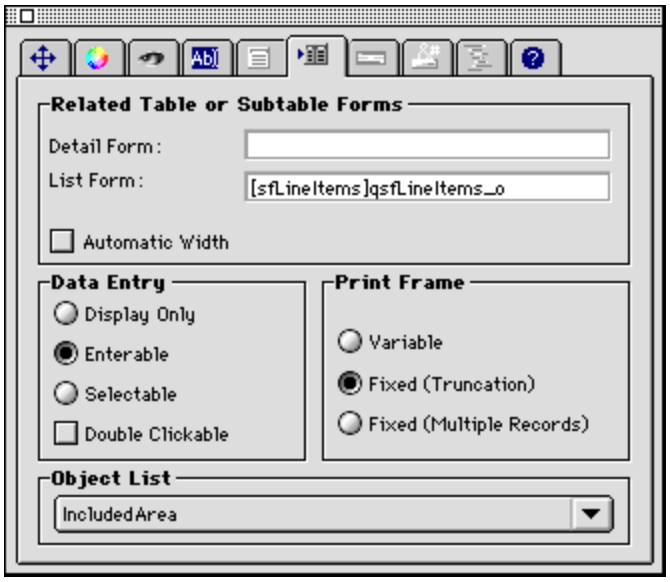
Enterable included forms

Line items appear on the second page of the invoice. These are not the line item records stored in the datafile, but copies of them temporarily stored in the sfLineItems subtable. The subtable is displayed on the invoice form using an included form object.

To use an included form, first create the form in the usual manner using the Forms Editor. The sfLineItems_o form is stored with the subtable.



Next draw an included form object on the form using the subform tool. The sfLineItems_o form is assigned to the object by click-dragging the outline of the form from the Explorer onto the object on the layout. You cannot directly enter the name of the form in the List Form area.



The objects on the subform are set to be enterable, and are given scripts so that they will respond to the values entered. The subform object, as it appears on the invoice, must also be set to be enterable.

Clairvoyant look-up

Clairvoyant look-up refers to the feature where the application looks up values based on partially entered specifications. In the invoice demo three fields support this type of data entry: the client name, item code, and item name fields. We'll only consider the client name look-up since the other clairvoyant fields behave in much the same way.

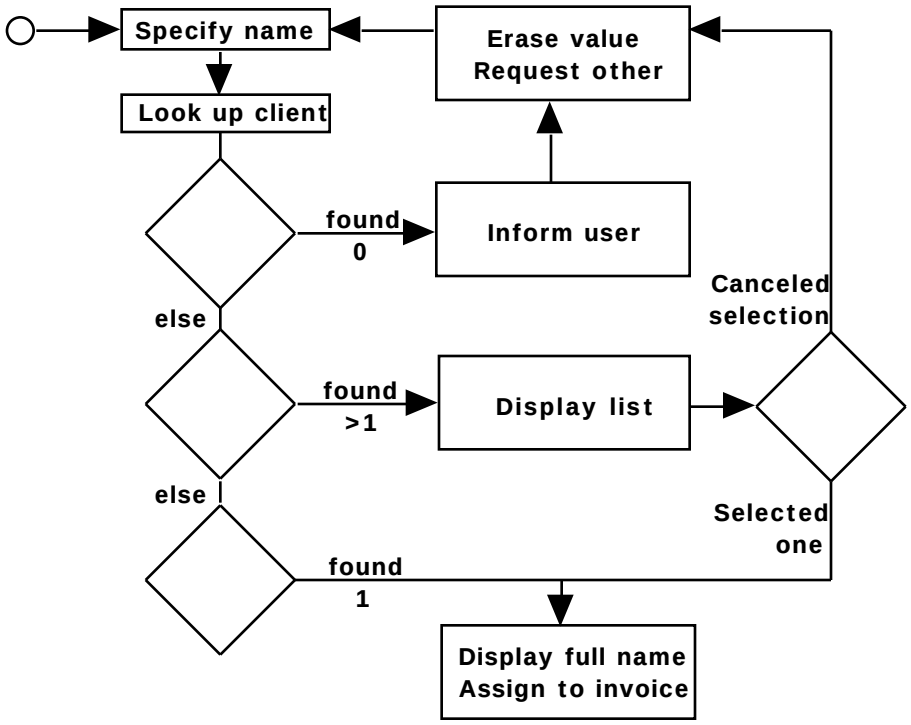
To use the client look-up, enter the first letters of a client's name in the client

area on the first page of the invoice. When you tab out of the field, the application examines the value entered. If the value does not end with the wild card character ("@"), the system adds it. Ending the search string with a "@" sign indicates to 4D's search engine that it should locate all records whose field values begin with the indicated letters.

When the application looks for a client it can either find none, one, or multiple client records. The look-up is performed in the ICAssignClient method. Each case is handled differently.

If...	Then...
No client found,	erase the name entered and inform the user.
Multiple clients found,	display list and ask user to pick one.
One client found,	display full name and assign to invoice.

The flow chart for this look-up process is shown below.



When the system identifies the client that the user intends to list on the invoice, it displays the client's full name in the client entry area. It also stores the ID of the indicated client in an invoice field.

When the client ID is stored with the invoice, it is referred to as a "foreign key." In general any record will store the foreign keys of all the records to which it is related. This foreign key maintains the link between the invoice and the client, not the client's name. That is to say, the client name field, in which the look-up was entered, is not actually stored with the invoice. Whenever an invoice is displayed or modified the name of the related client is looked up in the client table based on this unique client ID.

Process interaction

The "About" dialog box displays a simple animation that consists of the repeated display, in rapid succession, of a sequence of 15 pictures. This is an illustration of process interaction. The dialog box is displayed in one process while the other process runs in the background sending a regular stream of messages to the dialog process, telling it to display a different picture.

Two local processes are spawned by the SYAbout method. The dialog box is displayed in a process named "$AboutProcess," which is governed by the SYAboutProcess method. The background process which acts like a timer, runs in a process named "$AboutTimer." It is governed by the SYAboutBoxTimer method. These processes can only be local processes because they do not access the datafile. If they did any data access at all they would have to be redefined as global processes.

The $AboutTimer process runs a Repeat loop that first delays the timer process for a short time, issues the Call Process command with the ID of the $AboutProcess, then checks the state of the $AboutProcess process. This loop keeps repeating until it determines that the $AboutProcess process has terminated.

```
    `SYAboutBoxTimer
    `This method triggers an Outside Call event in the About process.
Repeat
  DELAY PROCESS(Current process ;5)
  CALL PROCESS(◊AboutProcessID)
  PROCESS PROPERTIES  (◊AboutProcessID;$Name;$State;$Time)
Until  (◊KillAboutTimer | ($State<0))
```

The Call Process command triggers an On Outside Call event directed to the dialog's form method. Every time the outside call event occurs the form method assigns the next picture in the sequence.

The commands used for interprocess communication in this example are

New process
PROCESS PROPERTIES
BRING TO FRONT
DELAY PROCESS
CALL PROCESS

Interprocess communication techniques are crucial to effective multiprocess applications. Processes run concurrently through a mechanism of preemptive multi-tasking at the level of the 4th Dimension application. Processes are not only used to support multiple windows, but also to support multiple concurrent processing.

In supporting multiple concurrent processes 4th Dimension goes well beyond simple multiple windowing. To see multiprocessing in action, open the About the Invoice Demo animated dialog box, then click back on the control screen. The animated about box continues to run in a separate window without interfering with your use of the invoice screens.

Summary

The invoice demo illustrates various 4D design methods and programming techniques. The demo tracks a single inventory level, and does not distinguish quantities on order from quantities remaining in stock. The system also does no accounting. Be aware that the accounting entries involved in invoicing are quite complex.

The purpose of the demo is to introduce you to many of the important elements in 4D that are useful in developing an invoicing system. The invoices provide a template for the construction of a more comprehensive invoicing system.

4th Quarter Accounting Solution, from which this demo was derived, is a complete, integrated and fully modifiable business and accounting platform. It is available as source code with full documentation, technical notes, and training materials.