

Omnis 5 version 1.2 2
New features and possibilities of the latest version of Omnis 5. Also a few cautions about changes in features documented and undocumented.

Sales Transactions 6
Before creating a billing (invoicing) system on the computer, it's important to understand the concepts involved and how business practices determine each system's needs. A (continuing) short course in accounting.

Computerized Invoicing 12
As in most database design, it's important to design for the information, not just to mimic the forms involved. Here we discuss the variations in information flow and File structure that you may encounter in designing billing or invoicing systems.

Multiple Main File Reports 18
Using procedures to define reports and a List to sort the records involved gives us almost unlimited capabilities. Here are some techniques for doing a seemingly impossible, but commonly needed task — sorting subrecords from one File in a single report.

Converting from 3 to 5, Part 1 24
By popular demand, this article outlines the problems you will face in converting existing Omnis 3 Plus applications to Omnis 5. It also gives some useful tips and techniques. The first in a series of articles.

Data Entry into Lists 30
These are very important techniques that rely on combinations of features and principles in Omnis 5. They have mystified many people who have encountered them on my example disks (where they were included without explanation), so here is the explanation. The two most useful techniques are given here.

Tips and Techniques 36
More clever algorithms and Omnis 5 hacking for your viewing pleasure. The definitive Age in Years expression and other useful trivia. Requests and suggestions are always welcome and will be properly credited when used.

Example Disk Program 39
The most complex simple disk so far in this series is available this issue. It thoroughly demonstrates the nine File example from the Multiple Main File Reports article and other useful tips. The price is a little higher than usual, but I'm sure you'll understand why.

Reviews, Benchmarks and Omnis 5

by David Swain

I am often asked why I seem to be the only software reviewer who gives Omnis 5 a positive nod when I write in the major magazines. I can only respond that I seem to be the only one who has spent enough time with the product to understand its true sophistication — and I also appear to be the only one to have worked on projects complex enough to *require* the advanced capabilities that only Omnis offers.



Figure 1 Schematic of a typical Omnis 5 review in any one of a number of Macintosh or PC trade journals.

If other reviewers have had any experience at all with a database product, it has either been with a “simple” product like File-Maker or with a strictly relational product like dBASE, Fox-BASE, or 4D.

For example, I wrote an Omnis vs. 4D “Point-Counterpoint” style article for the June '90 issue of MacUser. In private conversation, I learned that my counterpart, the 4D advocate, had never successfully created an application with more than five or six Files. He *had* been an Omnis developer in the early days of Omnis 3, but had “never been able to get it to work well”.

In fact, his early drafts seemed to be comparing 4D 2.0 to Omnis 3 (not even 3 Plus!) instead of Omnis 5. The article was a few months late while I tried to straighten out some of those misconceptions and blatant falsehoods. *continued on page 38*

Developer Haven Down Under

In March I attended an Omnis Developer Conference in Australia sponsored by Solutions, Ltd., the Australasian distributors of Omnis. It reminded me of the old days here in the States, so I thought I'd spend a few lines to share my impressions of the conference.

The most striking thing I saw there was the unabashed enthusiasm of the people attending. They were enthusiastic about Omnis 5, about the things they were learning at the conference, and about the impressive level of support they had been receiving from Solutions. There were also many worthwhile speakers there, including people from Apple Australia, the government, MacWorld Australia, a national software distributor, and Gavin Foster from Blyth UK. *continued on page 38*

Omnis 5 version 1.2

The latest version of Omnis 5 to be shipped as of this writing is numbered 1.21. It has a few new features (and a few new bugs!) that I feel you should be aware of.

First Impression

The first thing that most of you will notice are some changes to the standard menus. The wording and positioning of items in the standard Omnis 5 menus now conforms even better to the standards used in most Macintosh programs. This doesn't hurt anything on the Windows 3.0 side since no clear standards have been set there and those that are evolving generally follow the Macintosh model. Where there is no model to follow from other Macintosh programs, the wording has still been made more concise in most cases.

For example, instead of saying *Show design menu*, that File menu item now simply says *Design menu*, allowing the associated check mark to indicate whether that menu is to be shown or hidden. This applies to all similar items. The old item *Change user password* has been condensed to *Password*. And where we were once offered *Select application*, we are now more appropriately prompted to *Open application*.

In addition, a number of menu items have been put in more appropriate menus. *Change data file*, *Report destination*, and *Page setup* now appear in the File menu as they would in any other program. In addition to being able to *Open* a different application file, we also have the choice of *New application* and *Close application*.

Short and Long Menus

Perhaps the most radical change that arose with this version is the advent of the long and short menu modes. Long menu mode is very similar to the menu structure we have always had in the Omnis 5 development environment. Short menu mode is designed for the newcomer to Omnis or the person who just needs to get in and do simpler applications quickly.

The most significant change in the long menu mode is in the Design menu. Instead of a growing list of

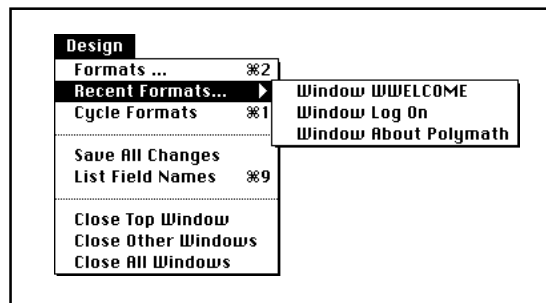


Figure 2 Long menu Design menu showing submenu of recent formats

up to nine of the most recently used formats, there is now a single line entitled *Recent formats...* that accesses a submenu of up to twenty most recently used formats (see Figure 2). Not only does this allow the programmer more flexibility when working with a complex set of formats, but it gives us the same "reach" when selecting any of the *Close window* options no matter how many or how few formats we have been working with.

Selecting *Short menus* from the long File menu puts us in Short Menu mode. The first thing we notice is that, in single user mode, both the Design and Utilities menus are automatically placed on the

menu bar for our use — even if they weren't there before. This is because we no longer have the option to install or remove any of the Omnis menus. This is the only change to the File menu in Short Menu mode.

The Design menu is another issue entirely, however. The entire top section (the one giving us access to various kinds of Format) has changed. There are now five items each giving us access to one of the five types of Format. Selecting any of these options opens a window that displays an alphanumeric list of the currently defined Formats of that type and offers buttons for creating, modifying, and otherwise manipulating a Format selected from that list.

The Window Formats and Report Formats windows have an additional button with exciting potential. This is a button that allows the programmer to create a default Window Format — or two default Report Formats — directly from a File Format (selected from a list of those available).

Unfortunately, the current version of Omnis doesn't give us a way to customize the attributes Omnis uses to build these windows and reports and the ones chosen by Blyth's programming staff are in almost total conflict with the way I have chosen to program Omnis. For example, I prefer to use 12 point Chicago font for field labels and 12 point Monaco for fields. Omnis uses 9 point Monaco for both and differentiates between labels and fields by using entry fields with a border. I rarely use the bordered entry fields in my work. The default procedures for these windows are designed for multi-window switching. I prefer a

very different programming technique and would have to completely rewrite all of these procedures. The reports use the Monaco font (including a number of bold text blocks) when I would much prefer Courier. As a consequence, I don't use this feature.

Another reason I don't use default windows or reports is that this feature is *only* available in Short Menus mode. Since I make frequent use of the *Recent formats...* option in Long Menus, I prefer to work there. There is no equivalent feature of the general Formats window in Long Menus mode. If only...

Window Manipulation

A number of new features and enhancements have been given to us for manipulating windows. Rather than dealing with these under separate headings of "variables", "procedure commands", and "window attributes", I think it makes more sense to discuss them as a group.

For people who deal with multiple window manipulation, there is a new Event Message variable, #TO-TOP. This is a Boolean value that will normally be detected at the Window Control level (see Vol I No 1, pp. 20-25) just as #AFTER is normally detected at the field procedure level. A "YES" value for this variable indicates that the current window has just been brought to the top, either by clicking on that window or by the *Open window* command. It also is shown in the sys(84) message string along with the name of the window that was brought to the top when such an event occurs.

We now have more options available for specifying the location and size of a window when we open it.

The *Open window* command now not only lets us dictate the position of the upper left corner of the named window, but the lower right one as well. This means that we can actually *resize* a window on opening it with this command in the form of

```
Open window { name/top/left/
             bottom/right }
```

The positioning parameters are measured in pixels from the top left corner of the screen. For example, if we want to open a window that has a title bar to the full size of the monitor, we can issue the command

```
Open window { name/20/0/
             [sys(104)]/[sys(105)] }
```

The value of 20 in the top position parameter is the height of the title bar in pixels. The *Open window* command positioning parameters ignore the size of the title bar.

In addition, we no longer have to struggle with bothersome algorithms in square bracket notation just to get the window centered on a general screen. There is now a simple optional parameter we can invoke called "center" ("centre" on some versions) that does the job. The option works fine, but it has its limitations.

The "center" parameter can be correctly interpreted by Omnis if entered in either lower or upper case. The only problem with this parameter is that many non-US versions of Omnis recognize "centre" as the correct spelling and trigger a devastating error when asked to interpret "center". Perhaps if Omnis could use one or the other...

There is also an option called "stack" that opens the named

window on a 20 pixel horizontal and vertical offset to the one currently on top — that is, if *that* window had been opened using the "stack" option. The first window to be opened using this option is placed in the upper left corner of the screen. The next window opened using this option is properly stacked on the first one *even if other windows had been opened in the meantime without this option*. Consider this procedure:

```
Open window {window 1/center}
Open window {window 2/stack}
Open window {window 3}
Open window {window 4/stack}
Open window {window 5/stack}
```

The first window would open centered as expected. The second would open in the upper left corner of the screen, *not* stacked on the first. The third would open wherever it had been left in modification mode. The fourth and fifth would be stacked in sequence based on the position of window 2.

The new parameters for *Open window* are mutually exclusive — that is, only one of them will actually work if you try to use more than one at a time. For example, trying to use both "stack" and "center" (to try to get the next window to stack on one that is centered) will cause a devastating error (one that closes all windows). Either "stack" or "center" can be used with the positioning parameters, but the positioning parameters take precedence in most cases. The top and left parameters will be used and the bottom and right parameters will be ignored. "Center" will be completely ignored when used with top and left. "Stack" will be ignored with regard to positioning the current window, but the next window opened with the "stack" parameter will be stacked

as if the first one had been placed in the upper left corner.

There remains no way that I can see to begin stacking windows anywhere other than in the upper left corner using these parameters. There is certainly no way to stack on a window that has been centered by that method. We *can* go back to our centering algorithm, though, and use a couple of temporary fields to control the stacking process. This method is outlined in the Tips & Techniques section of this issue.

Window Attributes

The fact that we can now specify all four corners of a window when opening it goes hand in hand with some of the new window attribute options available in version 1.21. If the operator doesn't like the size window we have programmed, we can give them the option of manually resizing it by including a grow box on that window. This is a control that resides in the lower right corner of a window and can be dragged by the operator to resize that window as needed.

Another new window sizing control is the zoom box. This is a box that resides at the right end of the title bar of a window. Clicking on this box opens the window to fill the entire monitor area. Clicking on it again returns that window to its original size.

We can also create a window that is the appropriate size for a large monitor, but still allow operators with smaller monitors to use it by including horizontal and vertical scrollbars on the window. These allow the operator to manually

show and work with areas that otherwise would be off the screen. During data entry, the window will shift automatically if the cursor is sent (by tabbing or shift-tabbing) to an area off screen. When creating windows that might be oversized for some monitors, the top/left/bottom/right method should be used for opening the window to keep the window boundaries inside the monitor area and ensure that the scroll bars will be available.

These window attributes are selected from the new Window Attributes window shown in Figure 3. They are *only* available for windows with a title bar. Dialog frame, simple frame, and no frame windows are not allowed to be resized.

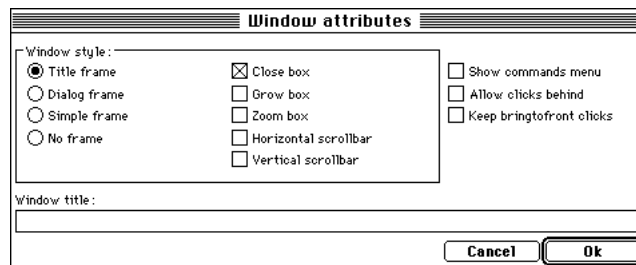


Figure 3 Window attributes window

Field Attributes

There are significant new attributes for a couple of window field types that are worth mentioning here. They are primarily user convenience features that you can provide for your operators. They won't affect your programming.

If you ever have trouble displaying all the columns you would like in a List Field, there is now a remedy. We have the option of providing a horizontal scrollbar on any List Field. This will allow the operator to scroll to information formatted in the List Field but out of view to the right or left. Unfortunately,

this will not scroll any titles or related entry fields you may have placed above or below the List Field, and we are still limited to twenty field names assigned to a List in this version of Omnis, but the horizontal scroll bar can still be a very useful feature.

We can also assign both horizontal and vertical scrollbars to picture fields in a Window Format. These bars can be assigned independently — that is, you can use one or the other or both as necessary. There is no apparent interaction between the selection of these scrollbars and the *Do not scale* option for picture fields. One would assume that this option should also be checked when checking scrollbars, since the bars would be totally useless if the displayed picture were scaled to the size of the picture field.

Port Control

There were no new procedure commands added to the 1.21 version of Omnis 5, nor were any removed or enhanced. One related feature *was* enhanced, however. We are now given a little more control over output to a port in the port setup dialog accessed by the *Prompt for destination* command (Figure 4). This dialog is found by first either executing the *Prompt for destination* command or choosing *Report destination* from the File menu, selecting "Port", and then clicking OK.

For a brief review, this dialog allows the operator to select and set up a port for print or telecommunications output. The dialog shown here is from the Macintosh version of Omnis 5, but the Windows version is similar. In the Windows version, the ports available are

selected from a list instead of radio buttons (because there are so many more choices). There are two more baud rates, 110 and 150, to allow for some of the more archaic equipment that can still be associated with some PCs. Finally, the checkbox at the bottom is labeled "Convert to ASCII" rather than "Convert for ImageWriter".

The additions to the port parameters we can specify are the horizontal and vertical pitch of the output — known in Omnis as "characters per inch" and "lines per inch". The important thing to know about these parameters is that they *do not send control characters* to the printer (or modem) to set up these specifications in the output device. Instead, they are used by Omnis to properly justify the output field values with padding spaces and to determine page breaks.

Color Table

The Omnis5.inf file has even less to do these days. One of the things it did in all previous versions of Omnis 5 was to store the custom color table set with the appropriate option of the Defaults menu. Unfortunately, this linked a custom set of color definitions with that copy of Omnis, not with a specific application. When the color table was changed, it was changed for all applications running with that copy of Omnis 5.

The color table is now stored in the application file. It can be seen (and even copied into other applications) in the Applications part of Utilities as a "font table" named #COLORS (along with the Window and Report font tables). No more will strange color combinations (other than those you have selected, of course) appear in your applications at your user sites.

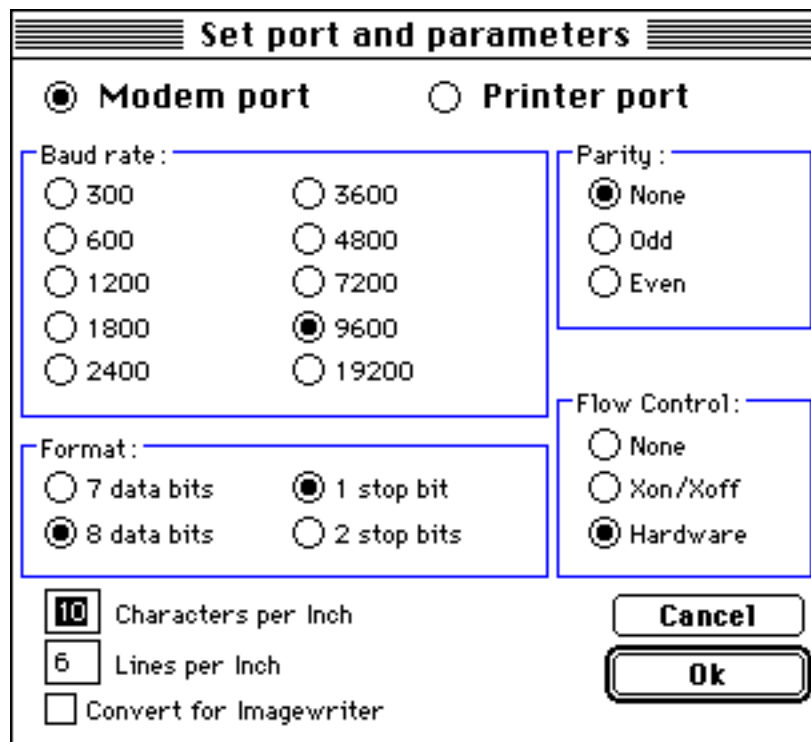


Figure 4 Port parameters dialog

Also, we can select the colors to be included in the color table from a standard Apple color picker on the Macintosh. This allows us to choose from sixteen million hue and saturation combinations instead of being limited to only 256 choices as we have been in the past.

Serialization

In an attempt to gain more control over the updating process at large multi-user sites, as well as to cut costs associated with increment-

ing users, Blyth spent a great many man hours perfecting an identification scheme where our serial numbers actually mean something. Here is how it works.

You initialize your copy of Omnis 5 as before, giving your name, business, and serial number when prompted the first time you launch the new version. Should you ever

need to add more users to the system, you simply call Blyth Software (or the distributor for your country) and give them your serial number, the number of users you need to add, and your credit card number (very important!). You will then be given another number for use with the Omnis 5 Upgrader program that will bump you to the number of users requested. Also, when new versions of Omnis 5 become available, you only need to use the Upgrader program supplied to trans-

fer your serial number and all extra users to the new version. No need to remember all those numbers.

This new "update protection" scheme only works with the new Omnis5.inf file distributed with your 1.2 update disk. Your original Omnis5.inf file is still valid and you can still use incrementer disks for adding users to your copy of Omnis. You cannot use both the Upgrader program and Incrementer disks — they are mutually exclusive.

Sales Transactions

The main income generating activity for most businesses is the sale of goods or services. Accurate and flexible tracking of these sales type transactions for a given business offers the owners or managers an opportunity to have their fingers on the pulse of that business. This can mean that those decision makers may have more accurate and timely information than their competitors on which to base their decisions for future courses of action for that business. Sometimes this competitive edge, or the lack thereof, can mean the difference between success and failure for a business.

In the physical realm of most businesses, the pieces of paper that are used to track this sort of transaction are called invoices, bills, sales slips, or receipts. Compiling information from these physical records can be a laborious task at best and has traditionally required a specialized labor force of accountants in larger businesses. Many small businesses don't even bother with these details — and suffer as a result.

In this article, I will examine the potential advantages for a business in being able to accurately track this information. Then, in the following article, I will detail the File structures and indicate some of the interface constructs we can use to build a database system in Omnis to flexibly manage and report on this kind of information flow.

Terminology

Since there are many other types of transactions that can occur within a business, the word "transaction" itself is a little too broad to

use in describing the sales tracking process. I prefer to use "invoice" or "invoicing system" as these terms are more descriptive of this class of transaction.

In the context of this discussion, when I use the word "product", I will be referring to anything tangible or intangible that is sold to a customer. The term "product" will also include services here, since it is simpler to speak of commodities of exchange in a general sense. The principles discussed here apply to housecleaning and gardening services just as they do to retail stores. The product of a rental business is the *use* of an article, although most rental stores also have a retail sales aspect.

A Game of Numbers

Numbers play a decisive role in the life of any business. Just as the laws of physics determine how the universe works, the laws of finance determine whether businesses grow or fail. As the saying goes, failure to recognize these laws in action does not make a business immune to their consequences. Only by careful observation of the raw numbers and certain key ratios derived from them can a business hope to make the choices necessary to survive in our increasingly competitive environment today.

Different types of business have different needs both in the information to be tracked and the methods used to capture that information. For example, some businesses deal in such volumes of commodities sold to faceless customers that tracking sales by customer would be a useless waste of time. I am speaking here of busi-

nesses like fast food outlets, grocery stores, movie theaters, and other such businesses. The raw numbers from the cash registers tied only to the products purchased (plus occasional input from special offer coupons redeemed and consumer surveys) are enough information on which to base most important decisions.

On the other hand, many businesses need to maintain close contact with each customer. They must make sure that the customer is made aware of special offers and new products as they become available and they use individual customer feedback (both direct and indirect) to determine what new products to carry and which old ones to discontinue.

Mass Market Needs

The numbers generated by sales of hamburgers, produce, or tickets are certainly significant to these large-scale, mass market businesses, but tying each sale back to a specific customer is unnecessary. These businesses thrive on television and newspaper advertising and mass merchandising. They do not have to be concerned with targeting specific customers in their bid for sales dollars. (It is becoming more feasible to track this information in the case of grocery stores with ATM paystations and laser barcode checkouts, however.)

If these businesses use mailing lists at all, they are usually bought from companies whose business it is to buy and sell such information. The mailings so generated are often no more personalized than to be addressed to "Our Neighbors at..." and are expected to have a fairly low rate of return. Such mailings saturate a geographic area and are the epitome of "junk mail".

These companies don't bother maintaining lists of potential customers — they deal with a broader, statistically-based picture and individuals are meaningless. Volume is all that matters.

The expansion of the SQL capability of Omnis has made working with such companies a much greater possibility for those of you who are independent consultants. It has also brought many people into the Omnis family who work within such companies. Both of these are reasons for including this discussion where I have always mentioned only customer-based invoicing systems before.

Mail Order Businesses

In the past, mail order companies have operated like mass market companies. They would buy mailing lists, send out catalogs, and wait for the orders and money to come rolling in.

Today, it is necessary to be much more competitive. There are simply many more mail order companies out there competing for not many more dollars. These businesses must target their customers much more carefully than ever before and they must also provide much more personalized service if they expect to retain their customer base. Repeat business is essential, but harder to maintain.

Many mail order companies have become much more specialized in their offerings. They offer high quality merchandise (for example, household goods) to a selective audience (people who want and can afford high quality household goods, but are too busy to go shopping for them). Customer retention and expansion through word-of-mouth advertising are of utmost

importance to the survival of such specialized mail order houses — but these can only be achieved through good customer service and repeated contact. Blind mailings have too small a return to warrant the expense.

Detailed and flexible information management can add strength to these businesses. If their volume is large enough, it is possible to create special mailings, and even special catalogs, for certain classes of customer based upon the types of items they have purchased in the past. This is done to ensure a higher rate of return on the advertising dollar.

Detailed customer purchase pattern tracking also allows these businesses to venture into other related areas when they are ready to expand. For example, a company that sells hiking and camping equipment can target people who buy certain types of gear as potential customers for trekking packages to exotic locations around the world — a venture into the travel business!

Many of you will have the opportunity to create systems for this type of business sometime in your career. There seem to be new ones popping up nearly every day. You might even be tempted to start your own — you will already know how to manage the information!

Narrow Market Needs

There are many traditional businesses that have always relied on a targeted customer base. These narrow market companies include manufacturers and wholesalers of specific product types (plumbing supply distributors, for example) who are limited to a relatively small number of customers who can af-

ford to buy the quantities of product that they must sell in order to be profitable. More often than not, these companies rely heavily on a sales force that makes periodic calls on each customer. These companies are only as effective as the information management skills of their field salespeople.

Besides narrow markets, these businesses also often have narrow margins. They must strive to remain price competitive while maintaining product quality and generating a profit. Many of these businesses turned to computer technology years ago to aid in the numeric juggling act they must perform to balance the quality-price-profit equation. But most are still using the same computer or computer service they started with years ago and are now losing the competitive edge they once had.

An added twist to systems for this type of business is that they often have to track bids or quotes in addition to sales. They may have also evolved some very complicated, but impressive discount structures in cutting special deals with important clients — in fact, they may in some cases be cutting their financial throats with these discounts and not know it.

Cost accounting is also an essential part of information systems for these businesses because of their narrow margins — but this is beyond the scope of our discussion in this issue.

The opportunities for consulting with this type of business are many and varied. You may be required to build a system for a local area network at the home office for data entry operators to log in field orders or you may need to build an SQL interface for the sales force to log

onto the company's mini computer remotely. In any event, these companies are used to dealing with contractors and are also used to paying the rates that you have to charge for your services. They may even be pleasantly surprised at how inexpensive your rates are relative to what they've been paying for maintenance on their mini!

Small Businesses

Local retail stores and service businesses are sometimes the poorest managed businesses with regard to information systems, yet they could each have such potential if the owners would learn how to play their numbers right. These businesses are often started by people with a special skill or interest who decide to open up a shop without any real idea of how to run a business. They rent business property, redecorate nicely, stock up, place a few ads in local newspapers, and wait for the money to come rolling in. Customer interest may be sparked initially because of the novelty of a new store opening, but this soon fades unless appropriate measures are taken to maintain consumer interest.

These business owners are not stupid people, but there are few places to go for the education they need to be successful. They pattern their activities after mass market companies, consciously or unconsciously, because that's the way they have heard that businesses are run. Their information systems are another matter.

They keep track of their customers by *remembering* their names and interests. Their customer list is most often a type- and hand-written list that they photocopy onto mailing labels — if they even go to the extreme of sending out an occa-

sional flyer. They manage their inventory by considering what they would like to see on their shelves and *remembering* what sold well last year at this time. Human memory does not often make an effective information management system for a business — even for a small one.

Most of these business owners eventually learn the rudiments of bookkeeping (often under duress), but they fail to keep up with the task until tax time (thinking they are only doing this for the government) and often never learn to interpret the numbers.

Paper systems cannot serve the needs of this type of business today. The job of compiling information from the paperwork is too much for an individual or a small staff when added to the other tasks they must perform daily. Many small business owners get sold a computer system, again because they have heard this is something they must have. But the off-the-shelf software sold to them by their computer dealer is usually either written with a different kind of business in mind or is too general to be tailored to their specific needs. And the software doesn't come with instructions on how to *use* the information it stores.

A custom tailored computer information management system can be invaluable to such businesses — especially if the consultant can also educate the business owner in the effective use of that system. It can allow for instant (or nearly instant) access to current sales and financial reports for faster decision-making (and better presentations to financial institutions for loan requests when needed).

This custom system can also greatly

enhance direct communications to customers to generate more sales and indirect communications from customers (in the form of sales figures) to help determine consumer demand. It can even be designed to allow for tracking of advertising responses so that the business owner can better manage that often hefty expense item!

By the way, these are the types of businesses you and I will most often be asked to help. They can be both the most frustrating and the most rewarding of clients. Frustrating because they often come to us as a last resort and don't understand why our service costs so much money. Rewarding because the combination of proper education and a good information management system, tailored to their specific business, can make such a positive impact on their business effectiveness.

One-Time Sales

There are many businesses that rely on repeat customers, but that sell their merchandise on a one-time basis. Included in this group are art galleries, antiques dealers, auction houses, pawn shops, and junk yards. The common element here is not the merchandise carried, but the fact that inventories are constantly changing and that inventory items are necessarily one-of-a-kind.

Most of these businesses will need to know who bought which items on a short-term basis for accounting purposes, but the long-term information of importance is the link between individual customers and the *types* of items they have bought. For an art gallery, this can include type of artwork (painting vs. sculpture vs. print), artist or style, and price range. For an auc-

tion house, this could be a combination of the types of articles purchased and the industry segment to which that customer belongs (production equipment for the bakery business, for example).

These types of business often sell their merchandise “on consignment”, meaning that they simply market items for other people without actually investing in the products themselves. Inventory management becomes a much more complex task in this case, since they are actually maintaining inventories for each of these suppliers and must communicate inventory levels, sales and commissions on a periodic basis to each of them.

Business Practices

Individual businesses each have their own practices regarding pricing and discount structures, shipping and other charges, and commissions and consignment rates. In addition, many businesses are obligated to collect sales, excise, and use taxes for their regional governments — and issues regarding what items are taxable, who is exempt, and how much to charge vary widely from place to place and from time to time.

The combinations of business practices and government impositions seem endless, but here are some of the more common elements seen in invoicing systems.

Special Pricing

Many businesses have more than one price for each item they sell, depending on who is buying the items or how many of them are bought. The most common variations of this practice are quantity price breaks and pricing based on customer category.

Pricing by quantity is not exactly the same thing as discounting, although it also results in a reduction in unit price when selling more units in a single transaction. Think of quantity pricing as being the “permanent” pricing structure and discounts as being a “temporary” or “transient” extra reduction.

Quantity pricing works like this. Let’s suppose that the base unit price of an item is \$6.00. This is the price used when selling a single unit in a transaction. If the customer buys six to eleven units of that item at one time, the unit price is reduced to \$5.75. A purchase of twelve or more units further reduces the unit price to \$5.65.

The quantities six and twelve are referred to as “price break” quantities. The purpose of this pricing method is to promote high sales volume on each invoice. This type of pricing is used by many retail and mail order businesses and most wholesale businesses. Mass marketers generally don’t build this strategy into their structures, but may offer coupon specials that mimic this practice.

Customer categories are also used to reduce prices for certain customers, usually those who have established certain minimum purchase criteria over some period of time. For example, a business might have three categories of customer labeled “retail”, “wholesale”, and “preferred”. Each product will then have three prices — the price used on an invoice is determined by the category in which that customer falls at the time of the invoice.

This pricing method is used to promote long-term high purchase volume and repeat business. Again, most wholesalers and manufactur-

ers employ this method to promote higher sales volume over the long term, but many retail and mail order businesses are beginning to use similar pricing strategies as are a number of multi-level marketing firms.

Some businesses may employ both of these schemes simultaneously with more break points and more categories.

The choice of strategy is driven by the perception, among the decision-makers of each business, of what incentives will promote the greatest long-term profit margin. Higher volume per transaction or per customer can significantly reduce administrative and sales costs. It can also lead to lower actual cost of goods when purchasing in greater volume from a supplier, who in turn lowers sales costs — and so on, right up the chain to the ultimate source! Accurate tracking of sales and cost information (and learning to use that information wisely) makes this system work to everyone’s advantage.

Discounts

Another incentive offered by many businesses to increase sales is the discount. There are a variety of methods and rationales used for discounting sales of goods and services — here are some of the most common.

Sometimes a discount is assigned to a specific product. This may be done to sell more of a slow-moving item or to get rid of inventory for a product that is to be discontinued.

When a discount is offered on a popular product, it is generally tied to the sale of something else. For example, a minimum purchase of other products may be required to

receive the discount on the popular product. In this case, the price reduction on the product that is selling well anyway is being used as an incentive to increase total sales volume.

A popular product may even be given away *free* (on a limited basis — and tied to sales volume) if such an incentive is expected to spark enough of an increase in sales to warrant the cost.

Discounts are sometimes applied to the overall invoice, while at other times they are applied on a line-by-line basis. In addition, the discount may be a specific dollar amount, or it may be given as a percentage of the invoice or line item total.

As you can imagine, this could lead to system-paralyzing complexity if you tried to allow for *all* possibilities in designing a generic sales tracking system. A custom system, limited to only those practices of a specific business, will always be much more efficient.

Discounts are also sometimes offered on certain items as a means of tracking advertising effectiveness. This is one of the primary purposes of coupon ads for mass market businesses. This is a separate issue that we will treat in a later OmniScience, but this kind of discount must still be reflected in the invoicing system like any other.

Sales Tax

Many types of business are required to collect taxes for their local or regional government. The most prevalent such tax is the sales tax. This tax has traditionally been levied on the consumer for the purchase of goods, but many governments are also imposing a sales tax on services as well.

The determination of what is or is not taxable varies from one government to the next and may change over time (generally with the list of taxable items getting longer). Merchants are required to collect sales tax from the consumer for each transaction and then summarize all sales made on a periodic basis (monthly, quarterly, or yearly), remitting the tax on the taxable portion of those sales to the government.

Businesses that accept telephone and mail orders and ship products to customers outside of their tax district often have an extra information burden. They may not have to charge sales tax on taxable items sent to a location outside their district, or they may have to charge a different amount because the destination either still falls under their same general governmental jurisdiction or because there is a reciprocity agreement between their government and that of the destination.

For example, in the State of California we have over ten distinct sales tax districts (generally county governments with their own add-on taxes), plus a basic sales tax for any other parts of the State. We are required to report sales made into *each* of these districts and deduct sales made to points outside the State. New districts are added from time to time and the percentage tax in each district can change periodically — sometimes even in the middle of a month! Other parts of the country have similar problems, and I'm sure that those of you outside the U.S. also have these headaches to contend with.

If sales tax is not collected, the merchant is still liable for the tax — and ignorance of the law is no excuse for non-compliance. It is

therefore extremely important for the merchant to be able to accurately collect and report on this revenue they manage for their government. Since that pastime is completely non-income-producing, it is also important that the process be done as quickly as possible without sacrificing accuracy.

The same principle applies to other forms of merchant-collected taxes. These can include, but are certainly not limited to, luxury tax, liquor tax, cigarette tax, etc. Each business has its own headaches here.

Shipping Charges

With shipping costs skyrocketing, more and more mail and telephone order businesses charge an additional amount on each invoice for shipping. This practice is designed to help recover the seller's costs for this service and pass some or all of it on to the consumer.

The alternative is to increase prices to cover the cost of shipping, but that tends to make the pricing seem less attractive and can have a long-term negative impact on both sales and profits.

Some businesses use a weight-based shipping charge in their invoicing system. Although this is undoubtedly the most accurate system for recovering shipping costs, it requires additional information overhead (knowing the weight of each item on the invoice plus the weight of packing materials). This method is most often used by businesses that sell heavy items where shipping costs can easily approach, or even exceed, the actual invoice total.

Most businesses include a nominal charge for "shipping and handling" that approximates the shipping

cost. The purpose here is to offset as much of the actual cost as possible without adding the extra overhead required for accuracy.

There are two prevalent methods for assigning a shipping charge figure. Both are based on the invoice total. The simplest (from the database designer's point of view) just takes a percentage of the invoice total amount as the charge. This can be set up to be calculated automatically — even to be rounded to the nearest 25¢ or whatever.

The other method relies on a "shipping charge table". This was the simplest method to use with paper-and-pencil systems (before the advent of the electronic calculator) because a clerk only had to find where the invoice total fell on a chart and use the corresponding charge. This method is still in wide use today, although the needs that caused its evolution are not as valid. It takes a little more cleverness to design a computerized system to emulate this process.

Receiving Payment

Selling goods and services is only half of the actual transaction. The other half is getting paid. There are a number of ways that payment can be made (and tracked) in an invoicing system.

Retail stores, restaurants, and mass marketing companies that deal directly with the public have the easiest time at this in many ways. These businesses are generally "cash-and-carry" operations — that is, people pay for their merchandise at the point of purchase. And many of them *only* accept cash in payment — no checks or credit cards. Money is given, merchandise and necessary change is returned. Simple and clean.

Credit card payments are treated like any other as far as invoices are concerned. The only additional information required is the credit card number, expiration date, and authorization code.

Mail order houses (and many other types of business) can have more problems to contend with in their sales information flow regarding payment. Sometimes in the real world, when a check or cash payment is received with an order, the customer might either send too much or too little. The problem is how to keep track of that discrepancy and eventually make it right.

Overpayments are dealt with by issuing the customer either a credit or a refund.

CODs and sales made on account are forms of delayed payment. Here merchandise leaves the point of sale with a *promise* of payment. Underpayments also fall into this category. We will explore accounts receivable in the next issue.

Commissions

Many businesses have salespeople who are paid a base salary plus a commission on each invoice they generate. It then becomes important to be able to track sales by salesperson and summarize this information on a periodic basis.

Consignments are usually dealt with in much the same way. In this case, though, we speak of the business keeping the commission on the item sold. A salesperson may also receive a portion of that commission.

Back to the Books

Ultimately, all information about the financial end of sales transac-

tions needs to find its way back into the general bookkeeping system for that business. (Refer to the discussions in Issue #2.) Entries must be made to the General Journal to reflect all sales activities logged in the invoicing system.

In some businesses, this can be done on a transaction by transaction basis. For most businesses, however, this is not a practical alternative — even for a computerized system. Whether we're talking about a mass market business or a small retail store, there would just be too many entries.

The standard alternative is to make a single journal entry from a periodic sales tally or summary of sales activity. This is usually done on a daily or weekly basis, often from cash register tapes or some similar source. This tally would include any breakdown of sales and revenues appropriate for that business (e.g., sales by product type).

Food for Thought

This article raises a number of issues that must be considered when designing sales tracking or invoicing systems. It is intended to set the stage for our design discussions in this and the remaining four issues of this volume of *Omni-Science*. Issue #4 will discuss Accounts Receivable and tracking payments received, Issue #5 will discuss Inventory Management (including receiving and warehouse movements), Issue #6 will discuss Customer Demographics, and Issue #7 will discuss Advertising Response Tracking.

Now that we have discussed the possibilities that abound in the world of sales, let's look at how we can model these sales practices using *Omnis*.

Computerized Invoicing

In a basic sales transaction a product or number of products is sold to a specific customer on a particular day for a specific price. In setting up a computerized model of such transactions for a specific business, our File structure depends upon the information needs and business practices of our client's enterprise.

A major key to setting up a file structure that can track this kind of information very flexibly is to determine whether more than one product can be sold at one time. (This most often is the case.) It is also important to determine whether, and how, our client needs to track customers related to each sale. Finally, we must recognize that the ultimate purpose of the invoicing system is not to generate a piece of paper called an invoice (although this is usually necessary in some form), but is to give us an opportunity to track sales by customer, sales by product, and sales by date range, etc. — and as many combinations of these as are appropriate for our client's business.

Single File Invoices

In some special circumstances, an invoicing system can get by with only two Files: a Customer File and an Invoice File. This is the case where a business only sells one product (or one product at a time from a limited and unchanging list of possibilities). An example might be a cleaning service that only charges for a number of hours at a fixed rate on each bill.

Although businesses operating on such a simple level are very rare, they do exist and their needs give us a starting point in exploring invoicing data structures.

The basic Invoice File needs the following fields. First, a *unique reference number* should be used for identifying an invoice for communications ranging from simple questions to disputes and collection proceedings. The *date* of the transaction is next in importance, along with the *total amount* of the transaction.

In the special case of a single line invoice, a *description* of the transaction is also important. This description may, however, be spread over several fields, depending on the nature of the business and the need for tracking different kinds of transaction.

A description like "for services rendered" is too vague for any kind of tracking. A description like "three hours of general housecleaning at \$50.00 per hour" is much more realistic. This description would be spread over three fields: the *product type* (general housecleaning, hourly rate), *quantity* (three), and *unit price* (\$50.00). Sometimes it may be necessary to separate the units from the product type — in this simple example, it was not.

For further convenience, the bulk of this information may be derived from the customer record itself if a contractual arrangement has been made. We would simply have fields in the customer record for the *current unit price* and, perhaps, the *expected quantity*. On starting a new invoice for a customer, our procedure would use these values as defaults. The values would still have to be stored in each invoice record, since our contract could change from year to year. We may also occasionally have a need to override these defaults under spe-

cial circumstances (cleaning up after a large party, for example).

Finally, if "general housecleaning, hourly rate" were our only product, it would only have to be stored in the report format. No need to store the same string over and over again!

Mass Market Sales

These are sales where it is important to keep very careful track of sales by item from a large list of products, and where many items can be purchased in a single transaction, but where we either can't or don't need to keep track of which customer bought which items. The texture of the sales portion of the data is the focus for the decision-maker here.

The transaction is the primary element in this system, but we can't do a good job of flexibly tracking sales by item with only a large Invoice File with a quantity and unit price field for each product we could conceivably sell. We can gain flexibility by having a separate File for our products.

Since a sales transaction in this business might include a sale of more than one product — and since we hope to sell each product on more than one invoice — a Many-to-Many relationship exists between our Invoice File and our Product File. A linkage File is needed to properly track this relationship (see Vol I No 1 p13 and Vol I No 2 p5). I generally use the name Line Items File for the linkage File in an invoicing system.

The basic fields in the Invoice File would still be *invoice number*, *date*, and *total amount*. The detailed description of the transaction is now spread over the other Files in the structure.

The description of each product to be sold is now stored in that products record in the Product or Inventory File. Each product should have its own unique *code number* and a general *description* or name of the product. If there are different sizes of a product, or some other important kind of subset, each size will usually require its own code number. The size may be included in the verbose description or *size* may be a separate field. The *current unit price* may also appear in this File to be used as a default during the invoicing process.

Mass market businesses are still concerned about customer information, but only in the *type* of customer or their *profile*. Demographic information on the customer can be included in the Invoice Header File in these cases, since individual customers don't have to be tracked.

General Structure

The two business types mentioned above are special cases. Most businesses need to both keep a close watch on what products are selling (or not) *and* keep in close communication with their *individual* customers to remain competitive. The invoicing systems we build for them must allow for this.

The four File invoicing structure (Customer, Invoice Header, Inventory or Product, and Line Items) has been the subject of many of my earlier writings, so I won't go into the need for each of these Files further here. Instead, I will elaborate on the fields required in each File and the variations of this mix of fields that may be dictated by different business practices. I will also discuss other Files that can come into play in the overall invoicing application.

Demographics

Issue #6 will deal with this subject in greater detail, but I will give you the basics here. For some businesses, it is enough to maintain a simple mailing list of customers with their names and addresses (sometimes even their telephone numbers). But many astute businesses owners and managers have learned that their products or services appeal primarily to a small subset of humanity. They also realize that a key method of expanding their business is to locate more potential customers and concentrate most of their sales efforts on those people.

Tracking demographic information in the Customer File can help build a *profile* of a typical customer for that business. Mailing lists of other people who fit various demographic profiles are readily available from many sources. If a business owner can determine what type of person frequents that business, there is a good chance that new customers can be found in a very efficient cost-effective, and inobtrusive way. Only people with a high probability of wanting or needing the goods or services of that business will be contacted, so other people won't have to deal with unwanted mail and the business won't waste money and natural resources in the prospecting effort.

The demographic information of importance will vary from business to business, but can include such fields as gender, age, income range, home ownership, specific hobbies or special interests, profession, education level, distance travelled to patronize the business, etc.

For many businesses, knowledge of the customer profile can be their most valuable asset.

Invoice Header Fields

The basic invoice header record requires a unique invoice, receipt, or transaction number, a date, a total amount, the method of payment used, and some detail about that payment (for example, a check number or Visa or MasterCard number and expiration date). If the business also has to deal with sales or use tax, an appropriate field must be supplied for that figure as well. (More on taxes later.)

For internal purposes of the business, it may also be necessary to *categorize* the invoice in some ways. For example, some businesses may want to distinguish between telephone order, mail order and walk-in sales. Other businesses may want to track sales that were generated by specific ads or in response to specific mailings. Still others may need to track time of day or day of the week to determine peak and slack sales periods. Any such categorization will require a field in the Invoice Header File to capture the necessary information.

These category fields may sometimes hold information derived from other fields. For example, day of the week could be derived from the invoice date. Although redundant data storage is looked down upon by database academicians, an application can often be made to operate more efficiently in the real world with a few extra fields. In this case, a single-byte integer field can be used to store the day of the week (derived using the *dtw()* function) *so the value doesn't have to be calculated each time it is used*.

In today's world, time is a more costly commodity than disk storage — and the value of time continues to climb while the cost of storage is dropping rapidly. Anything we can

do to save time over the life of an application is certainly worth a few bytes per record!

There is another important point about real world database systems that can be made in our Invoice Header discussion. Often it is necessary to store information that would at first glance seem to be redundant information from further up the Connection hierarchy. For example, I store the Customer Name and Address information in the Invoice Header record. This takes up quite a bit of storage space, but there is a very good, practical reason for doing so.

The information in a record higher up the hierarchy may need to change at some point in time. If it were not stored redundantly in the Invoice Header File, future reports could print out erroneous information. If a customer moves and the address in the Customer File were relied upon for the Ship To address for the Invoice, that address would now be incorrect.

This might not seem to be a serious problem, but consider a sales tax audit. If a customer who had lived out of state a few years ago moved to an address down the block from the business, and only the current address of the customer were available in the invoicing system, a sales tax audit would indicate that the business is liable for sales tax on all old invoices for that customer in prior years (not to mention interest and penalties) *based on the company's own records.*

We as system designers are charged with maintaining the accuracy of our client's data over time. We must constantly be on guard for things that could change in the real world and make that data stored in the computer seem wrong. Storing the

Ship To address in the Invoice Header record is one of those things. Doing this, however, also has the added advantage of allowing the operator to override the default address and perform drop shipments for a customer.

Another "category" that might be included in the Invoice Header record is one to track sales by a particular salesperson or data entry by a particular operator as a measure of productivity. In the case of sales by salesperson, a Salesperson or Employee File might be used, with Invoice Header record connected to that File.

Alternatively, a productivity check on data entry operators could be done in the background by employing a user record and a personal password at log-on. The User record would then be a source of the operator's initials (or some other identifier) for marking that Invoice. It would not be necessary to connect the Invoice to the User record here.

Other financial information might also be included in the Invoice Header record. I will explain more about accounts receivable in the next issue, however I will point out here that accounts receivable information could be included in the Invoice Header record if selling things on account or with terms is the general practice of the business being modeled. If sales are only occasionally done on account, it makes more sense to have a separate Accounts Receivable File.

Shipping Charges

The Invoice Header record can also be used for tracking certain types of charges and credits tied to the overall Invoice (rather than tied to a specific Line Item). An example of this is a shipping charge. A nu-

meric field for the shipping charge amount might be included in the Invoice Header File if business practices call for one. While we're at it, a field indicating shipping method (First Class mail, surface freight, next day or second day delivery, will call, etc.) could be included, as well as a field indicating whether the order had been shipped. The ShippingMethod field would best be cast as an Integer field using Radio Button fields for display and manipulation. The Shipped/Unshipped indicator would be a Boolean field.

That indicator is an example of what is called a "flag" field. There could potentially be many other flags associated with an Invoice record. For example, we may want a flag that states whether or not an Invoice has been voided. We don't want to simply delete the Invoice record and all of its line items because we need to know what happened to each invoice number (auditors are fussy about these things). That number would already have been generated within the system. Another flag might indicate whether an invoice has been paid — and if so, we might need field to show by what method and on what date. An indexed Boolean flag indicating whether the invoice had been paid gives us direct and immediate access to all unpaid invoices — an important group of records!

Line Items Fields

All of this information gathered into the Invoice Header record is great for tracking the overall sales performance within a company and purchases by a specific customer, but this is just the tip of the iceberg. The real heart of the sales system depends on what happens with specific products on a line item basis within these invoices.

The Line Item record will generally only hold a single price — the *unit price* at which the product sold in this transaction. This field and a field for the *quantity sold*, are the two basic fields in any Line Items File. If costs vary over time (they usually do) and if the *unit cost* at the time of sale can be accurately known, a field for this figure is also useful in the Line Items File.

Pricing Practices

Many businesses have more than one price for their products. The two most common multiple pricing schemes are *quantity* pricing and *customer-type* pricing.

Quantity pricing means that for different ranges of quantity sold, a different unit price is used. The quantity at which the price changes is called the *price break* point. To implement this, we need to know how many ranges the business uses. For our example, we will assume three ranges and, therefore, two price break points. Our Product File will then have two Price Break fields and three Unit Price fields. Our procedure for generating the default unit price for a Line Item record will then check the Line Item Quantity field to assign the Unit Price value — like this.

```
If IT_QUANTITY<PR_QUANTITY_A
  Calculate IT_UNITPRICE as
    PR_PRICE_A
Else if IT_QUANTITY<PR_QUANTITY_B
  Calculate IT_UNITPRICE as
    PR_PRICE_B
Else
  Calculate IT_UNITPRICE as
    PR_PRICE_C
End if
```

Customer-type pricing bases the unit price on a category field value in the Customer record. The Product File will have a Price field for

each possible Customer Type value. For example, if we only recognize “Regular” and “Preferred” customers, and we have a Boolean field to distinguish them, our pricing might be done with a simple expression.

```
CU_PREFERRED*PR_PRICEP+
  not (CU_PREFERRED) *PR_PRICER
```

For more complex categorizations, a Character or a Boolean field would be used for Customer Type with the added complexity that brings to such expressions.

Discounts

Many businesses give an allowance or discount on some invoices as a sales incentive. Here are some ways of handling the more common discounting practices.

The simplest method for us to implement is the discount given on the overall Invoice once the subtotal has been calculated. This is the method used when customers redeem coupons, for example. We would simply add a field for this discount amount to the Invoice Header File and include a field on the invoicing window for the operator to enter a value. This type of discount is usually given on a *dollar amount* basis. The grand total for the invoice is then calculated by adding the invoice subtotal, sales tax, shipping charges, etc. and then subtracting the discount amount.

A related discount type is a *percentage* discount given on the overall invoice. This is most often given based on the *class* to which the customer belongs. For example, “preferred” customers may qualify for a 10% discount. The class information is retrieved from the Customer record and can vary over time, so a field for the class *at the time of sale* should be included in

the Invoice record. Similarly, the *current* discount rate for that class would come from the System Constants File, but that can also vary with time, so a field in the Invoice File should also be included for this information. Since the discount amount can be calculated directly from these values, it is not necessary to also include a field it.

Some businesses may prefer to offer discounts on a more individualized basis. In that case, the discount rate will be supplied directly from the Customer record. Otherwise, the discounting is dealt with as in the preceding paragraph.

Another common practice is to give discounts on products or services on a *line-by-line* basis. These may be *either* dollar amount or percentage depending on the practice used. If a percentage discount is used, both the computer screen and the printed invoice should also display either the discount amount or the resulting net unit price calculated from the percentage entered. (Only the percentage discount and the original unit price would need to be stored.) The procedures used to generate the subtotal amount (the one before taxes and other charges) simply needs to accommodate the line item discounting method being used.

Sales Tax

Another charge often included on invoices is sales tax as a percentage of the invoice total. Some businesses aren’t required to charge sales tax. A few states have no sales tax (although their numbers are decreasing rapidly) and sales tax is not a universal, international concept, but most business systems have a requirement of this sort. The variations in sales tax requirements can also be quite complex.

Merchants in many states are still blessed with a *uniform* sales tax for the entire state. Information systems for businesses in such states only need to tack on the additional percentage (for point-of-sale systems) or to track in-state sales vs. out-of-state sales (for mail order systems) to do a thorough job of accounting for sales tax. But increasingly many states are beginning to subdivide into various tax districts that have different sales tax rates. Some are entering into reciprocal agreements with neighboring states for purposes of assuring sales tax revenue. This only serves to complicate matters for the merchant and the computer programmer. The more complex the problem, the more varied are the possible solutions. Let's take a look at the more common and workable.

For applications to be run in states where there is a uniform sales tax, the easiest and most flexible method is to have a field for the state sales tax rate in the System Constants File. In a point-of-sale system, this sales tax rate would be used for every invoice or receipt generated. And in mail order systems, the sales tax would be applied to sales to customers within that state and would be zero for sales to customers out of state. Since the sales tax could still vary over time, it would be a good idea to include a field for the current tax rate in the Invoice Header File and transfer the tax rate value to each new record from the Constants File. To determine the sales tax for a specific Invoice record, a simple calculation using a Boolean subsection would suffice. The Boolean sub-expression would be numerically zero if the Ship To State did not equal the state code in the System Constants record.

```
(IV_STATE=SY_STATE)*IV_TOTAL*IV_TAX_RATE
```

Some customers may be tax exempt. They may be exempt institutions or they may be businesses that hold a valid sales tax license and are purchasing for resale. This should be tracked by the system. The best way to handle this is to have a flag in the Customer File indicating tax exempt status. This value would be transferred to an equivalent field in the Invoice record (since that status could change in the Customer record over time). Our sales tax expression would then be extended to multiply the tax status as well as the in-state status against the Total.

The case for states with multiple -tax districts or who have reciprocal sales tax agreements with other states is far more complex. Point-of-sale systems would not have to handle this, since all sales are assumed to be in the local district, but mail order businesses are generally required to collect the proper sales tax for the district *to which the shipment is being made*.

The method recommended here is to add a new File to the system for the Sales Tax District. The Tax District File would contain three fields — one for Tax District Code (two or three characters), one for the Name of that sales tax district (for printing out on reports and for displaying on the screen), and one for the Tax Rate in that district.

The File structure we will use here is based on the *permanence* of the record relationships. We connect Invoice to Tax District because, for all time, a particular Invoice will belong to that Tax District (even though tax rates may change). Invoice is also connected to Customer because, for all time, that Invoice belongs to the Customer, even if that customer eventually dies, becomes inactive, or whatever.

It is *not* good practice in Omnis to *connect* the Customer to the Tax District. If the Customer were to move to another Tax District, there would be an inconsistency in the data structure for all previous Invoice records. Instead, we should use a traditional *relational reference*, placing a field in the Customer File to refer to the Tax District by its code. This is the *current* Tax District for the Customer and is used to *look up* the proper Tax District record when a new Invoice is being created. At that point in time, the Tax District will be the same for Customer and Invoice, and the Invoice will be permanently linked to the Tax District.

In many states that require sales tax to be collected, there are exempt product classes. If a business sells both taxable and tax-exempt items, the distinction must be made at the Line Item level. A Taxable Total field should be included in the Invoice Header. The value of this field is used to generate the final Sales Tax Amount. The *current* Taxable status of an item is stored in the Product File, but the Taxable status *at the time of sale* is stored in the Line Item record, since even this can change over time. (In California, snack foods became taxable on July 15, 1991.)

Journal Entries

An invoicing system is more complete if it links into the basic bookkeeping system. Ideally, a computerized business system should automatically carry out *all* implications of *each* entry — so all the bookkeeping implications (Inventory updates and Journal entries) of an invoice are handled when that Invoice record is entered.

The Journal entry has a minimum of two Detail lines — a Debit entry

for the revenues generated (in this simple case, to Cash) and one or more Credit entries detailing sales by type of product. Sales tax collected also generates a Credit.

Many businesses have separate Ledger accounts for sales by category. For example, I track consulting income, training income, and sales of books and newsletters. An indicator field is included in the Product File to categorize each item for this process. This field is set up to use either a Character field to hold an alphanumeric code or a radio button field to store an integer value for each product type.

If Line Item records for an Invoice have been entered using a List, it is a simple matter to retrieve the subtotal for each product type and store each in a temporary variable set aside for that purpose. If we have three categories and have used an Integer field for storing Product Type, we might use #30, #31, and #32 for their respective subtotals using the *totc()* function as follows.

```
Calculate #30 as
  totc((PR_TYPE=0)*IT_QTY*IT_PRICE)
Calculate #31 as
  totc((PR_TYPE=1)*IT_QTY*IT_PRICE)
Calculate #32 as
  totc((PR_TYPE=2)*IT_QTY*IT_PRICE)
```

Our subroutine for generating a Journal entry would then look like:

```
Begin reversible block
  Set main file {Invoice Header}
End reversible block
Clear selected files
  {Invoice Header,Ledger,Line Items}
Calculate JO_DATE as IV_DATE
Calculate JO_DESCRIPTION as
  con('Invoice #',IV_NUMBER)
Single file find on SY_SEQNO
  (Exact match) {0}
Prepare for insert w current values
Calculate SY_LAST_ENTRY as
```

```
  SY_LAST_ENTRY+1
Calculate JO_NUMBER as
  jst(SY_LAST_ENTRY,'-5P0')
Update files
Set main file {Line Items}
Single file find on LE_ACCT_NUMBER
  (Exact match) {'100'}
Calculate DE_AMOUNT as
  IV_TOTAL+IV_SALES_TAX
Prepare for insert w current values
Calculate LE_BALANCE as
  LE_BALANCE+DE_AMOUNT
Update files
If #30
  Single file find LE_ACCT_NUMBER
    (Exact match) {'400'}
  Calculate DE_AMOUNT as -#30
  Prepare for insert current values
  Calculate LE_BALANCE as
    LE_BALANCE+DE_AMOUNT
  Update files
End if
If #31
  Single file find LE_ACCT_NUMBER
    (Exact match) {'410'}
  Calculate DE_AMOUNT as -#31
  Prepare for insert current values
  Calculate LE_BALANCE as
    LE_BALANCE+DE_AMOUNT
  Update files
End if
If #32
  Single file find LE_ACCT_NUMBER
    (Exact match) {'420'}
  Calculate DE_AMOUNT as -#32
  Prepare for insert current values
  Calculate LE_BALANCE as
    LE_BALANCE+DE_AMOUNT
  Update files
End if
If IV_SALES_TAX
  Single file find LE_ACCT_NUMBER
    (Exact match) {'700'}
  Calculate DE_AMOUNT as
    -IV_SALES_TAX
  Prepare for insert current values
  Calculate LE_BALANCE as
    LE_BALANCE+DE_AMOUNT
  Update files
End if
```

Notice that the Detail records are created in standard display order

(Debits first) and that they are all independent (each has its own *If* statement). If a thorough job is done in setting up this procedure, the Journal entry will *always* balance because all revenue and income source possibilities have been covered. The computer won't make arithmetic or copying mistakes!

Account numbers are hard coded here, but we could have set up fields in System Constants to hold those values just as easily. Each Ledger record is read into the CRB based on those hard coded values. It is not necessary to reread the Journal record in because it was just created and is still current.

For business that sell "large ticket" items and have only a few transactions per day, generating a Journal entry for each transaction is probably the best path to follow. But for businesses that handle hundreds or thousands of transactions per day, this would unnecessarily complicate the Journal and waste a lot of disk space. It is only necessary to make a single Journal entry summarizing an entire day's sales.

To do this, we would add another File to our system — a Daily Sales Tally File. This File has fields for accumulating revenues and income by category as well as other activity fields (number of transactions, etc.) that the business manager might wish to track. At the beginning of each business day, a procedure is run to transfer this information into a balanced Journal entry in a similar way to the procedure listed above. This procedure also creates a new Daily Sales record to begin the accumulation for the current day.

I admit that this discussion has been too brief, but I hope it has given you some useful direction.

Multiple Main File Reports

We saw in issue #1 that we can bypass the Main File - Connected File sorting limitations of the Report Generator by using an Omnis List as a sort buffer and manipulating the CRB (through a procedure) to generate reports sorted on many levels of a Connection hierarchy. The example given still relied on a single "Main File" at the base of this hierarchy for which we simply manipulated the *flat file image* of each record as it became current. It is possible to extend this method to create reports that have multiple "Main Files" ... But what exactly do I mean by that?

Example Problem

Consider the following scenario. We are building an application for the service department of a client. A service call (in this example) is always on a specific type of product for a specific customer. So our structure would contain a Service Call File connected to a Customer File and a Product File — a simple Many-to-Many relationship.

But there is more information to track. For each service call, our client wants to keep track of the problems that were dealt with on that call, the parts that were used to fix the problems (if any), and the contact person or persons associated with that call at their customer's site. We need to provide a means of reporting calls for a specific problem code, a specific part number, or a specific contact person for a range of service call

dates. We also need to be able to list service calls by Customer or by Product (or by both in either order), and in that listing show the contact people for each call alphabetically, the problem codes alphanumerically, and the part numbers alphanumerically for each service call.

Because of this need for flexibility, both in the number of record linkages to be stored and in the number of ways to retrieve subsets of records by indexes, it was determined that the application should use the structure shown in Figure 5. What was (originally) a simple Many-to-Many relationship has now become a maze of additional

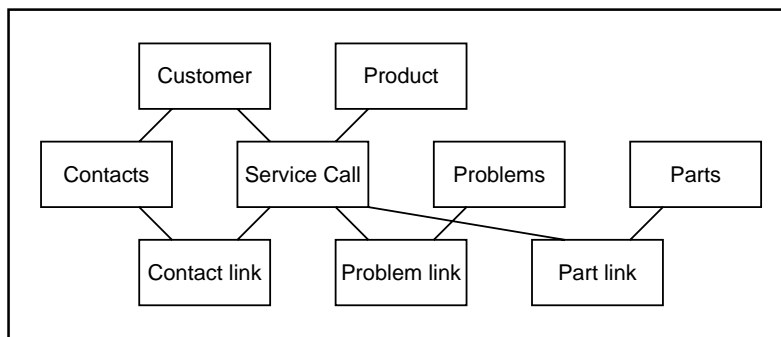


Figure 5 Service call with three Many-to-Many linkages

linkages to other Files from the first linkage File (the Service Call File). Furthermore, the Contact File has a legitimate Connection to the Customer File since each contact person belongs to only one customer site.

What Is The Main File?

We will assume that the data entry methods have been worked out (they are available on the Example Disk for this issue) and concentrate on the procedures required to generate the report showing the

"sub-lists" of contact people, problem types, and replacement parts for each service call.

In printing a list of Service Call records sorted by Customer and/or Product information, our first guess is that Service Call would be the Main File. This is perfectly true with regard to the File that we must use to determine which Service Call records to include in the report. But we have to go to the next level down our Connection hierarchy to gather the linkage information — *and there are three Files down there!* Each has to become the Main File in its turn for us to gather Contact, Problem, or Parts records associated with a specific Service Call record. And we are required to sort the records we find in each group as well!

It looks like we need to again use a List to help us in our sorting process, but how can we juggle three Main Files when we go to print our report from that List? And just what *are* we sorting anyway?

In addition to the File Format fields

we need to sort the report, we also need a variable to act as a switch that indicates which of our three linkage Files is the current Main File for each line of the List. For this example, we will use #20 as the Main File indicator variable.

The List we use as our sort buffer will then include the fields listed in Figure 6. Notice that we include all the fields on which the report will be sorted as well as the sequence fields for *each* of the linkage Files *and* the Service Call File. Each of these field values will be

required as circumstances change during the process of generating this complex report.

Let's now examine the steps required to print out the most basic of these reports — one sorted by customer, product, and service call date, that also prints all three sublists for each service call.

Gathering records

The Main File in our procedure for *locating* Service Call records to be included in the report will be the Service Call File. The search or index range criteria for selecting these records will only involve Customer, Product, and Service Call (usually date range) information. For each selection option required by our client, we will determine the optimized method of locating the necessary subset of records procedurally. Normally this would *not* be done by using the *Build list from file* command, but one record at a time in the CRB. Nothing is added to our List at this point.

For each record that meets our criteria, we then call a series of subroutines to collect each potential set of linked information for the current Call record. Each subroutine begins with a reversible block that sets up the environment for dealing with that one linkage File. The flat file image of that linkage File is confirmed and the sort field values for that flat file image are added to the List. Care is taken that the Files outside that direct-line hierarchy are cleared in the CRB. A diagram showing which Files are involved in each subroutine process is shown in Figure 7. Here is a typical record location subroutine.

```
Locate PBLs for CA
Begin reversible block
  Calculate #2D0 as CA_SEQNO
  Calculate #20D0 as 1
  Set main file {Problem link}
  Single file find on CA_SEQNO
    (Exact match)
End reversible block
Find on CA_SEQNO (Exact match)
If flag true
  Repeat
    Load connected records
      {Calls file}
    Add line to list
    Next (Exact match)
  Until flag false
```

1. Customer Name
2. Customer City
3. Product Code
4. Service Call Number
5. #20 (Main File Indicator)
6. Contact Last Name
7. Contact First Name
8. Problem Code
9. Part Number
10. Service Call Sequence No.
11. Contact Link Sequence No.
12. Problem Link Sequence No.
13. Part Link Sequence No.

For the report based on Product Code, sort field number 3 would move to the number 1 position. 1 and 2 would become 2 and 3.

Figure 6 Fields included in the list. The first nine are for sorting, the rest are for retrieving records for printing

```
Else
  Single file find on CA_SEQNO
    (Exact match) {#2}
  Load connected records
    {Calls file}
  Add line to list
End If
```

This procedure gathers all the Problem Link File records connected to the current Service Call record. Since Problem Link is the Main File, the associated Problem File records are also read into the CRB.

The use of the *Load connected records* command on the Service Call File ensures that the entire flat file image of the located Problem Link record is in the CRB when the sort information for that record is added to our List.

The subroutine begins with a reversible block that establishes the Main File as the Problem Link File and sets a value for #20 that corresponds with that Main File setting. Since we want the Contacts sublist to be printed first, followed by Problem Codes and then Part Numbers, we give #20 the values of 0, 1, or 2 when we set the Main File to Contact Link, Problem Link, or Parts Link respectively.

Remember that most commands performed in a reversible block set up values (and other states of the application) *for that procedure only*. Omnis reverts to its previous state with regard to these commands when the procedure finishes. So the Main File setting and the value of #20 will return to their previous values when processing returns to the procedure that called this subroutine.

The reversible block in this procedure also performs some “insurance” functions regarding the current Service Call record. It remembers the current Service Call record in two ways. The RSN is stored in #2 as insurance against the possibility of having to “manually” retrieve that record if there are no Problem Link records connected to it. That record is also “automatically” retrieved when processing returns to the procedure that called this subroutine because of the *Single file find* per-

formed on the Service Call records current sequence number value in the reversible block. This command doesn't change anything in the CRB, but it pushes the current Service Call record onto the State Stack (see Vol I No 1 pp 32-33) for automatic retrieval at the end of the subroutine.

The only way this subroutine will finish is when no more linkage records can be located for the current Service Call record. This is detected by the *Until flag false* criterion in the record location *Repeat* loop. The "flag false" indicates that the null record for the Main File (Problem Link) has been located. When this happens, we also end up with the null record in each of the Connected Files — in this case, Problem and Service Call. The automatic retrieval of the current Service Call record, set up as described in the preceding paragraph, is necessary so that the procedure that called this subroutine can continue on from the proper point in the Service Call File.

Even if there are no Problem Link records connected to a specific Service Call record, we would still like the report to show a message like "No Problem Codes associated with this Service Call" in the appropriate place, so we have also allowed for a *blank* line (with regard to Problem Link) to be added to our list of record sort images in that event. Remember that if the *Find on CA_SEQNO* with exact match fails, we again end up with the null record on the Problem Link, Problem, and Service Call Files. To have all the necessary information available for proper sorting of this "blank" record

image, we need to retrieve the Service Call record before adding the line of sort information to the List. This is the purpose of the procedure lines after the *Else* statement. We will examine how the report format deals with this special image later.

Sorting and Printing

The sort criteria for our List can be established in one of two ways.

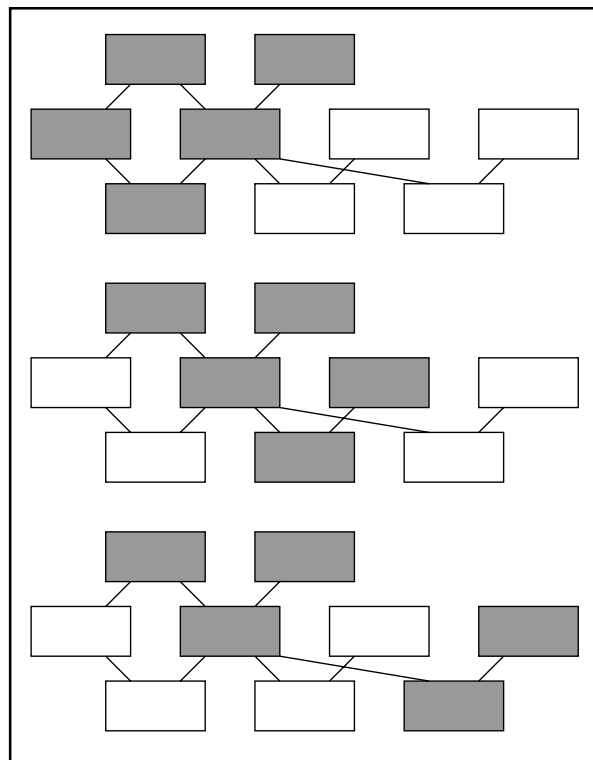


Figure 7 Files involved in the various flat file images of this example. Contact Link, Problem Link, and Part Link.

Selecting a report with a list of Sort Fields already set is the easiest way. The sort fields associated with the selected Report Format simply replace the current ones.

We can also procedurally set up sort fields using the *Set sort field* command. It is wise to precede your list of sort fields with a *Clear sort fields* command to clear the slate. If you don't, and if some sort fields already exist (from a previously selected report, etc.), you will be

adding to the sort fields instead of replacing them. After the *Clear*, the first use of *Set sort field* sets sort and subtotal level number one. The second use sets level number two, and so on.

Once the sort fields have been established, sorting our record images is very simple. The *Sort list* command sorts the current List on the sort levels we have set. The only requirement is that all of the sort fields *must* be in the definition of the List.

After sorting the List, our procedure must go back through that List from the beginning and re-establish the flat file image represented by each line. Once this has been done for a line, the *Print record* command is issued and the Report Template and Subtotal Tracker of the Report Format do their jobs. (Remember that the *Prepare for print* command must be issued before the first *Print record*.) The trick is to build the flat file image based on the right Main File.

We can use the value of #20 to make the choice. If #20 is 0, we will set the Main File to Contact Link and perform an exact match *Find on CNL_SEQNO* using the value in that field from the current line of the current List. Values of 1 and 2 would invoke similar blocks of commands for the Problem Link and Part Link Files respectively.

The *Find* command will also locate the corresponding records in the associated Many-to-Many File (Contact, Problem, or Part) and the Service Call File. If the *Find* fails, this indicates that the field with the linkage File sequence

number contained a 0. In this event, the Service Call record needs to be retrieved by using the Service Call Sequence Number value stored in the current line of the current List. Once there is a Service Call record in the CRB, the associated Customer and Product records can be read in using the *Load connected records* command on the Service Call File. The *Print record* command can then be issued.

The segment of our procedure that follows the loading of the List is shown below. The sort fields were already established by selecting the Report Format in the reversible block of the main procedure.

```
Working message (Repeat count)
    {Sorting}
Sort list
Close working message
Prompt for destination
Prompt for page setup
Prepare for print
Calculate #L as 1
Repeat
    Calculate #20D0 as 1st(#20)
    If not(#20)
        Set main file {Contact link}
        Find on CNL_SEQNO (Exact match)
            {1st(CNL_SEQNO)}
    Else If #20=1
        Set main file {Problem link}
        Find on PBL_SEQNO (Exact match)
            {1st(PBL_SEQNO)}
    Else If #20=2
        Set main file {Parts link}
        Find on PAL_SEQNO (Exact match)
            {1st(PAL_SEQNO)}
    End If
    If flag false
        Single file find on CA_NUMBER
        (Exact match) {1st(CA_NUMBER)}
    End If
    Load connected records
        {Calls file}
    Print record
    Calculate #L as #L+1
Until #L>#LN
Print totals
```

The Report Format

Although our procedure takes the place of the Record Sorter of the Report Format, the Report Template and the Subtotal Tracker are still very active. All programming elements of the Report Format are operational — but many of these perform in subtle ways that we must understand before we can make this reporting method work as freely as we would like.

The Record Section

The core of any Omnis 5 report is the Record Section. The *Print record* command might better be called the *Print record section* command, since that is exactly what it does.

As with the record location commands (*Find*, *Next*, *Previous*, etc.) and the *Prepare for edit* command, *Print record* also rereads the Connected records for the current Main File record when it executes. In general, this is a reassuring multi-user feature of Omnis — but for this report, it causes us a problem.

The problem is that we want to print some information in the report for each record image *even when the Main File record slot is empty*. Even though our procedure reads the Service Call record back into the CRB when the *Find on the Main File* fails, the *Print record* command *will read the null record for Service Call back in as a direct result of printing the Record Section for a null record in our current potential Main File*. This would seem to prevent us from printing a text block like

```
No parts used on Call #[CA_NUMBER]
```

If we need information from a Connected File record in the Record

Section itself when printing *this kind* of report, we must include a field in the Record Section that rereads the proper Connected File record. This field must be *Calculated* and *Autofind*.

In the example above, the field would be for CA_NUMBER and would be invisible, since the actual value is to be shown in square bracket notation in a text block. The calculation would be 1st(CA_NUMBER) since the only place the proper Service Call number exists reliably is in the current line of the current List. *The field must also be placed somewhere to the left of the text block* since fields are executed from left to right and the value in the text block depends upon this operation being performed.

If no Connected File information is needed for the “blank” Main File records, these Autofind fields are not required. However, we may still have similar needs in either the Subtotal Level or the Subtotal Heading Sections even if we don’t have them here.

Another feature that we must give our report is the ability to print alternative and/or optional lines in the Record Section (and, perhaps, other Sections). In this report, we will want to print different Record Section contents for each of our three Main Files, and alternate contents if that Main File slot is currently empty (e.g., if there are no Part Link records for a specific Service Call). This is a total of *six* possible, mutually exclusive formats for our Record Section.

I will emphasize here that it is *not appropriate or useful* to have more than one Record Section per Report Format. We can give our single Record Section completely different looks depending on circumstances.

We can change the look of the Record Section by simply suppressing some lines and printing others. Our report will have at least six lines in the Record Section — one for each of our possible Record Section formats. (Any of these formats could have more than one line, if necessary.)

To do this, we begin each line with an invisible field that determines whether or not that line should be printed. The field name used will generally be that of a temporary numeric field (I usually use #50 for this). The field will be declared *Calculated* and *No line if empty*.

The calculation will be a mathematical description of the circumstances under which that line should be printed. That is, if the calculation evaluates to a non-zero number (a Boolean “YES”), the line will print. If it evaluates to exactly zero (a Boolean “NO”), the line will be suppressed. The Main File setting is determined by the value of #20. The alternate lines for that Main File are printed based on whether or not a record is in the CRB for that File — that is, whether or not there is a non-zero value for that File’s RSN. For example, the line (or lines) to be printed if the Main File is Parts Link would use

```
#20=2&PAL_SEQNO
```

The line (or lines) to be printed to show that no parts were used on a service call would use

```
#20=2&not(PAL_SEQNO)
```

The Record Section would look like the one shown in Figure 8. It might

look a little complicated at first, but the resulting report is very easy for the reader to follow. Just remember — only one of these six lines prints per sorted record image.

Subtotals & Headings

The Subtotal Level sections print whenever the value of the corresponding sort level field changes. Also, all lower levels are assumed to change when a higher level changes. (That is, if level 2 changes, it is assumed that levels 3 through 9 have also changed.)

There is a slight difference in the way that the subtotal level sections relate to the CRB when printing one record at a time from a procedure as opposed to following

```
Record #5000: No contacts noted for this call
Record #5000: [CN_LAST_NAME], [CN_FIRST_NAME] CN_PHONE
Record #5000: No problems specified for this call
Record #5000: PB CODE PB DESCRIPTION
Record #5000: No parts needed for this call
Record #5000: PA CODE PA DESCRIPTION
Subtotals level 5
```

Figure 8 Record Section showing six mutually exclusive alternative lines

the *Print report* command. When *Print report* is issued, the Subtotal Tracker can anticipate subtotal breaks because of a communication with the Record Sorter. The last record in the previous subtotal grouping remains in the CRB when the Subtotal Section prints, and the next record comes into the CRB when the following Subtotal Heading prints. Field values from non-totalled fields are then available from appropriate records when these sections print. (Totalled fields contain the appropriate *subtotal* value in Subtotal Sections.)

When we print a report using the *Print record* command repeatedly, the Subtotal Tracker can’t know

that a subtotal break is required *until the triggering value changes* — and this can only happen *when the next record comes into the CRB*. This could pose a slight problem for us if we want to use CRB information from the Main File or a Connected File (usually for labeling purposes) in a Subtotal Section because our procedure would have already configured the CRB for the *next* record.

Recent versions of Omnis 5 have addressed this problem by maintaining a buffer with the last CRB configuration used during the printing process. This configuration is *reread* into the CRB when the printing of a Subtotal Section is triggered. The problem would seem to be solved.

Unfortunately, this condition persists *through* the printing of the following Subtotal Heading, and the next record *doesn’t reappear in the CRB until the Record Section is printed*. This means

that the Subtotal Heading information will be out of synch with the records in the following subtotal grouping — making the information in our report *very* wrong.

The answer, again, is to use *Invisible*, *Calculated*, *Autofind* fields to reconfigure the parts of the CRB that no longer conform to our needs. This is done within the lowest level (highest number) Subtotal Section that affects the values to be shown in the Subtotal Heading, and it is done after any necessary field values from those Files have been printed in the Subtotal Section (in left-to-right, top-to-bottom order). The calculation expression to be used would be of the form

```
lst(fieldname)
```

We use the field value from the *current* line of the List because #L has already been incremented by our procedure. The field name to be used must be for a uniquely indexed field for absolute accuracy. If we are not sorting on such a field, we need to add it to our List definition.

There is one other problem to be dealt with in this process. That is, there is currently no equivalent to the *Load connected records* command inside Report Formats. We must reconfigure the CRB for *each* File all the way up our hierarchy. Again, if a uniquely indexed field is not already in the List definition for each File on which we must perform this operation, we need to include one.

Subtotal Headings

We don't have different Subtotal Heading Sections for the various sort levels as we do for Subtotal Sections. There is only one Subtotal Heading Section and it is printed directly after the highest level (lowest number) Subtotal Section whenever a subtotal is triggered. In reports like this one, we would probably like to print different information depending on which Subtotal Section just printed — and which Main File applies to the next subtotal grouping.

The Subtotal Heading is can be made to react to the value of #20 just like the Record Section. When we print the first record for a new Service Call (the first Contact Link or a "blank" one), we would like to print overall information about that Call, including the Customer and

Product information. We won't want to print those lines again until first Contact Link record for the *next* comes up. At the same time we print these lines, we might also want to print heading information for the Contact listing for the current Service Call. When Problem Link becomes the Main File, we won't need to print the Service Call, Customer, or Product information again, but we might well want a new heading for our list of Problem Codes and Descriptions. The list of Parts used on this Service Call would get the same treatment.

Since the value of #20 has already been tripped to the appropriate value for the next subtotal grouping, we can use it to trigger or suppress the printing of various lines in the Subtotal Heading Section. The Service Call, Customer, and Product information lines, as well as the heading line for Con-

like those we developed for the Record Section.

Limitations

The limitations of this method are relatively few. As with any other report in Omnis, we are limited to nine sort levels. One of our sort levels here needs to deal with the lowest level Main File selection (the #20 value).

Another limitation — and this one could cause some serious problems in large applications — is that our report is limited to the number of records that can be held in a List. The maximum here in the current version of Omnis 5 is 30,000, but available RAM (due to total available memory, use of inits, MultiFinder, System 7 partitions, and use of other Lists) may limit the number of record images for this process still further.

A third limitation is that a List in Omnis 5 can only be defined with a maximum of twenty field names. If we are reporting on many levels of a Connection Hierarchy and must reconfigure the CRB on many fields that are not also used for sorting, we may reach our limit on field names before satisfying all our needs. This is a serious limitation of the current version of Omnis.

Within these limits, this method allows us tremendous reporting flexibility. If this process seems complex, you are correctly dealing with reality. Just take it a step at a time and go over each section of this article to understand the principles there. Then put the pieces together. I think you'll be pleased with the results!

Subtotal heading	
#50D0	CU NAME
#50D0	CU ADDRESS1
#50D0	CU ADDRESS2
#50D0	[CU_CITY], [CU_STATE] [CU_POSTAL_CODE]
#50D0	Call #
#50D0	CA NUM CA TYPE PR_CODE
#50D0	Contact Phone:
#50D0	Problem Code and Description
#50D0	Parts:
#50D0	Record

Figure 9 Subtotal Heading Section showing alternate heading lines for different subtotal groupings

tacts, would be set to print only if #20 equals zero. A value of one in #20 would cause the Problems heading line to print, and a value of two would trigger the printing of the Parts heading line.

An example of a Subtotal Heading Section for our report is shown in Figure 9. Note the fields for #50. They are *Calculated, Invisible, and No line if empty* with calculations

Converting 3 to 5, Part 1

Omnis 3 evolved over a long period of time and many developers and end users still have a lot of time and energy tied up in Omnis 3 applications. Omnis 5 offers some tremendous new features that could improve these applications — and Omnis 5 has a feature to convert Omnis 3 applications to the closest Omnis 5 equivalent. But the complete conversion path from 3 to 5 is not a direct one, and the decision of whether to convert existing Omnis 3 applications or to build completely new ones in Omnis 5 has been the subject of much debate since Omnis 5 was first released.

By popular demand from OmniScience readers, I am writing this series of articles to help those who have chosen to convert. This article is only an introduction to some of the basic problems involved in conversion and the solutions that I use. A second and, perhaps, a third article will be required (in the next issue or two) to complete these basic explanations. If enough interest is shown, I may publish an entire booklet on the subject to give more guidelines, suggestions, and examples.

To Convert or Not...

There are two basic camps in the 3-to-5 Conversion Controversy. The first group maintains that Omnis 5 is so different from Omnis 3 that it is useless to even try converting and existing Omnis 3 applications should just be rewritten from scratch. The other camp says that a lot of time went into building the old application and the facility is there to convert at least some of it to Omnis 5, so let's take advantage of that facility and save some time.

My position is closest to that of the second group. I very early made the observation that you will certainly want to rethink your application in the light of Omnis 5 methods, but the converted pieces that Omnis 5 can give you make great building blocks on which to base your new version. I feel that those who see Omnis 5 as completely different from Omnis 3 Plus are only looking at the surface (independent windows with font and graphic capabilities) and are missing the fact that Omnis still manipulates data in the same way.

The answer to the question of whether to convert depends on how good the Omnis 3 application was in the first place. A lot of those I've seen should be rebuilt from the ground up anyway — even if Omnis 5 had never evolved! But if the original Omnis 3 application is sound and functional, I maintain that you can save at least 30-40% of the time of a complete rewrite by starting the rewrite with a conversion and then picking up the pieces.

File Formats

An Omnis 3 application with a good, flexible File structure should retain the very same structure when converted to Omnis 5. The way that Omnis manipulates data hasn't changed beyond the fact that we can now have more than twelve Files open simultaneously. (Twelve Files was the limit of the CRB in an Entry Layout in Omnis 3. The limit is now sixty for both the Datafile and the CRB — assuming you have enough RAM!) The fabric of Connections that you may have woven will remain intact and will be perfectly useable — assuming that the structure was logically sound to

begin with.

File Formats are still used for exactly the same thing they were in Omnis 3 Plus — defining the structure of the data to be manipulated by the application. They convert directly, including their Format names. The most I usually need to do with converted File Formats is to change their names to match my current conventions, but there's no rush (or need) to do so.

Field Types

The field types from Omnis 3 Plus File Formats all have direct counterparts in Omnis 5 and convert appropriately to Omnis 5 File Format fields. This single fact convinced me that it is worth using conversion. If you have a sound File structure, you can at least save the time required to retype all your field names and assign data types to them. I eventually get around to changing the field names to match my current conventions, but again this is not necessary.

Some features involving fields *will* require special attention or alteration once the conversion has been completed. For example, we now have the facility to have long text fields and no longer have the facility to have linked string fields on entry windows. To get the same effect that we had using linked string fields in Omnis 3 Plus in a single scrolling long text field in the converted application, a little extra manipulation is required — especially if you also have a converted datafile with values in these fields.

After conversion of the application and a reorganization of the datafile (which removes the fixed length nature of string fields), expand the first of the linked fields to a maxi-

mum length large enough to accommodate all the characters in a completely full linked block. For example, if you have ten sixty character fields in the block, expand the first field to at least a maximum length of 600 characters.

Next, write a procedure to concatenate all the field values in the block (in the proper order) with an additional space between each one for padding. You can get fancier, if you like, and include code that only concatenates a field to the block if there is actually something in it to avoid adding a lot of unnecessary space characters at the end of the string. It is even possible to include a return character (chr(13)) to skip blank lines when more text follows on further lines.

After running the procedure, use a window with a scrolling entry or display field to check a representative sample of records to see that the entire string value correctly transferred to your single field. If everything checks out, delete or redefine the other fields in the block and reorganize again. All that remains to be done is to convert the field used to display this fields value to a scrolling entry or display field on each window where one appears and delete the fields that used to display the rest of the linked block.

Indexes

All indexes will be converted properly except for extended indexes (those with an index length longer than the field length, for example, Last Name extended over First Name) from Omnis 3. Extended indexes will be reduced to the maximum length of that indexed field.

At first glance, this seems like a total loss of a useful feature from

Omnis 3 Plus, but it is not possible to have this feature in Omnis 5. The ability to have extended indexes in Omnis 3 Plus was a by-product of having fixed length fields. In Omnis 5, all our string fields are now of variable length. This new feature designed to save disk space and “modernize” Omnis 5 also makes it impossible to have that old feature. Fortunately, it can be emulated without much effort when it’s really needed.

A string field in the CRB in Omnis 3 Plus would contain the characters in that string and enough null characters (chr(0)) to fill the entire length of the field. When an index value was created, Omnis would simply lift the value to be used from the CRB starting at the first character of the indexed field and taking the number of fields specified in the File Format for that index. If the index length were longer than the field length, no problem — the next field in File Format field number order was also adjacent to that indexed field in the CRB and a portion (or all) of its value would be used as well. The index sorts properly because a null character is the lowest in sorting order of any ASCII character.

Suppose we have a Last Name field fifteen characters long with an index length of twenty five. It is followed by a First Name field that is at least ten characters long. A collection of values in the Last Name index might look like this (dashes are used to represent the null characters:

```
John-----Sonny-----
Johnson-----Al-----
Johnson-----Benny-----
Jones-----Ken-----
```

The CRB in Omnis 5 works a little differently. String fields are of

variable length. A delimiting character is used to separate field values. Lifting extended values here would be almost like concatenating the First Name to the Last Name — and there are many cases where the resulting index would not be in the order we need. We also have no means of typing in null characters (even if the data entry operator knew how many to type in) and trailing spaces entered into a string field are trimmed as the cursor leaves that field. (Spaces already existing in the field — that were either imported or calculated — remain, however.)

We can emulate our old extended index by creating an extra field that contains a fixed length composite of the involved field values. We can build and maintain those composite index field values using the *jst()* function. For example, we might use the command

```
Calculate LAST_FIRST as
      jst(LAST_NAME,15,FIRST_NAME,10)
```

Although the *jst()* function pads with trailing *spaces* instead of null characters, these index values will still sort correctly. This is because the space character is the lowest in sorting order of any of the *printable* characters in the ASCII set. All characters that are lower are *control* characters.

We’ll have to discuss *using* this composite index at some other time. Even in Omnis 3 Plus it took some doing to allow for an exact match prompted find on a composite Last Name-First Name index value.

Connections

The Omnis Connection facility hasn’t changed in Omnis 5 and connected File structures convert properly with no problems.

File Modes

All Files from the Omnis 3 Library are converted to *Read/Write Files* in Omnis 5. If a File had been used as a “dummy” File in the Omnis 3 Library (one that is just used for its CRB slots and never has a record written to disk), its mode should be changed to *Memory-only File*. This is a more appropriate mode for that File given its use.

Since I write all my applications for the possibility that they might be used in a multiuser environment, my personal style requires that I change the default mode of all other Files to *Read Only*. When I want to write a new record for one of these Files, or to edit the contents of an existing record, I set only those that need read/write access to *Read/Write* mode. This is most often done in a reversible block to make this a temporary mode for those Files.

The one disadvantage to setting all Files to a default mode of *Read Only* is that a File has to be in *Read/Write* mode to import records — even when importing through Utilities. Omnis doesn't give any kind of warning that the File isn't in the proper mode until you get all the field names lined up and finally hit the Start button to set the import in motion. When you go away to set the File mode and then return, you have to start all over again arranging the field names in the required import order.

To avoid the problem, I include two extra steps in my process for importing. I first execute a procedure from my STARTUP menu that sets the appropriate File to *Read/Write* mode. I keep this procedure outside of the range available for the pull-down portion of the menu — usually at procedure number 50 — and

execute it with command-E. I then perform the import in the normal manner. When finished, I execute another procedure (number 51) that sets the mode back to *Read Only*. Since I rarely have to perform this task, the partial clumsiness of this process isn't much of a bother.

String Variables

Anywhere one of the standard string variables, #S1-#S5, were used in the Omnis 3 Plus library, they should continue to perform the same function properly in the conversion to Omnis 5. All the variables convert with no errors and they are each the same size or larger in character capacity.

A limitation on the use of these variables in Omnis 3 was that they were intimately associated with the Array. This is no longer the case (Lists have no intrinsic fields), so these can be used anywhere in an Omnis 5 application without worrying about the values seeming to change because of some process happening in the background. In converting Array structures to Lists, it is likely that you will want to use fields other than these temporary strings. I will give you some guidelines in the section on converting the Array.

Another common use made of temporary string variables was in semicolon notation in calculated message fields to cause the information displayed there to change if necessary when the screen redraws. This functionality converts properly to Omnis 5, but the temporary strings aren't really needed for this anymore. I just remove the temporary field name from such fields and let the field be a No Name field. This will avoid any conflict there might be with other uses of the temporary strings.

If the message field were used as part of an emulated radio button group in Omnis 3, the use of the temporary can again be eliminated. I will discuss radio button conversion later in this series.

Numeric Variables

The Temporary Numeric Variables, #1-#60, are now floating point SANE numbers in Omnis 5. In practical terms, this means that the order of magnitude of the number won't change if the number of decimal places is specified differently for the same variable within the same window, report, or procedure (as it would in Omnis 3) — but the value is not rounded off to that specified number of decimal places, either. Rounding errors can occur in reports where a value is derived and totaled in one of these fields because the full sixteen significant digits are being totaled — not the (rounded) displayed value. The `rnd()` function should be used to correct for this error in human precision in such cases.

Omnis 5 will occasionally make conversion errors on these variables. The error takes the form of not using the correct number of decimal places in the converted field or procedure command. Temporary numeric fields in windows and reports, or used as the target of a *Calculate* command, should be checked for decimal place accuracy after conversion.

Global Strings

The long Global String Variables, #G1-#G3, no longer exist in Omnis 5. They only made a brief appearance in Omnis 3.3. Anywhere one of these variables is encountered during conversion, the Unknown Variable (#???) is substituted. #??? has no data type and no length. It

is only a place holder indicating that a field not belonging to this application had been used in a specific spot.

There were two basic reasons to use a #Gx variable in Omnis 3.3. The most common was to avoid using one of the #Sx variables, since they were all associated with the Array. A less common reason was if the programmer needed a string variable longer than 79 characters. Very rarely, the #Gx variables would be invoked if the programmer needed more than five temporary strings.

If the programmer had used the #Gx variables simply to avoid using the Array Temporary Strings, and the values to be placed there will never be longer than 80 characters, an appropriate #Sx variable can be substituted for #??? in the converted application. (Remember to properly document your use of all variables so you don't get confused!) In any case, the programmer can now build any number of temporary string variables (each with a length of up to 32000 characters) in a Memory Only File and substitute one of those for #???

To help in tracking down where #Gx variables were used in your Omnis 3.3 library, you might want to use the Omnis Cross Reference Tool. This can help you zero in on the places where you need to substitute a new temporary for a specific #Gx.

Functions

All functions convert correctly. There are no changes in any Omnis 3 Plus function. We did receive many new functions in Omnis 5, though, that we might want to substitute for some of the ones we used in Omnis 3 for greater efficiency

and speed. More on this in a minute.

Expressions

Omnis 5 evaluates expressions according to the standard algebraic hierarchy conventions most of us learned in high school. Omnis 3 evaluates expressions on a strictly left-to-right basis, as though it were a hand calculator. If the conversion routines built into Omnis 5 didn't adjust for this difference, most of the expressions in our converted application wouldn't yield correct results.

Fortunately, Omnis 5 goes to great lengths to see that our converted expressions will evaluate correctly. Parentheses are strategically added to enforce the previous left-to-right evaluation order when it might be overridden by the Omnis 5 order of precedence among operators. Original parentheses are left intact.

There is one problem with this conversion method: we can get too many parentheses and lose the whole expression. For example, we might have this Omnis 3 expression

```
#1+#2+#3+#4+#5+#6+#7+#8
```

It will convert (see Vol. I No. 2, p. 23 for more on this) to

```
(((((#1+#2)+#3)+#4)+#5)+#6)+#7)+#8
```

Notice that six nested levels of parentheses have been used here — the maximum that Omnis 5 allows. *Expressions requiring more than six nested levels of parentheses in their converted form don't convert.* You will end up with a blank where the expression should have been. The converted expressions will often need to be slightly reworked for efficiency as well.

(Again, see Vol. I No. 2 and the associated Examples Disk for explanations and demonstrations of the lack of efficiency in expressions with unnecessary parentheses.)

Semicolon Notation

This feature of Omnis 3 has now become Square Bracket Notation in Omnis 5 and has gained a great deal of power and flexibility. All field references in semicolon notation in an Omnis 3 Library will be converted to square bracket notation and will work successfully in Omnis 5. In addition, the field names will now be tokenized, since square bracket notation now encloses "live" expressions.

Some of the places where you used semicolon notation — in message fields on Entry Layout screens or in No Name fields in reports — you may now want to consider transferring the strings that incorporate square bracket notation values to text blocks. This can both cut down on the number of fields you are using (an important consideration on a window) and improve readability of that information in modification mode (since the contents of No Name fields don't display unless that fields attributes are being modified).

On windows, you would only use this alternative if the square bracket value is to be evaluated as the window opens and never change while the window is open. This is because there is no way in the current version of Omnis 5 to redraw the window and have text blocks re-evaluate.

On reports, this isn't a problem because both fields and text blocks are evaluated each time they are put into the printing stream.

Entry Layouts

When Omnis 5 converts an Omnis 3 Plus Entry Layout, we end up with a lot of individual components for our converted application. Each of these will need a little reworking to fit our growing set of Omnis 5 conventions, but most of the rework is pretty simple. I will focus on some of the simpler objects on the window this time. Next time I'll discuss what we can do with the procedures and the more complicated objects (like Array leftovers).

Message Fields

Message fields no longer exist in Omnis 5. The most common use for Message fields in Omnis 3 was as field labels on entry screens. They were also likened to No Name fields in Omnis 3 Report Formats. Omnis 5 converts them to No Name fields on Window Formats. The Chicago font is retained by these fields.

The most appropriate conversion followup, from the standpoint of common usage, would be to turn Message fields into text objects on Omnis 5 windows. However, if you did not follow the guideline in "Unlocking Omnis 3 Plus" about giving Message field labels higher field numbers than data entry fields, this might significantly alter field number order on the window, which could ruin any converted EDWC procedures that relied on those field numbers.

The method is simple. First, open a Message field by double-clicking on it with the selection tool and copy or cut the Lookup table string from the field's definition. Then select the text tool, click somewhere to create a text block, and paste the string. The resulting text block will retain the same font and other attributes of the original No Name

field because Omnis remembers the attributes of the last selected object, including font, size, foreground and background color, line style, fill pattern, and justification.

This can actually lead to a small problem with some fields. If the field is right or center justified, or if it is very long, the paste cannot be completed without an intermediate step. The current version of Omnis 5 can only paste (or accept keyboard input) into a text block that is big enough to accept the full width of the string being entered. For some unexplained reason, text blocks begun with right or center justification already selected are much shorter than left justified ones. Also, right and center justified text blocks retain their size when being edited while short left justified ones (that have not been drag-sized) open up to accommodate a few more characters. Text blocks of any size can be made larger (or smaller) by dragging one of their corner handles. These difficulties will probably be cleared up in future versions of Omnis.

Grid Placement

Omnis 5 doesn't seem to understand the concept of "leading" when converting fields from Omnis 3 Entry Layouts to Omnis 5 windows. The 12 point Monaco and Chicago fields (which have a total height of 15 points including the leading) overlap fields on adjacent lines by 3 points after conversion. They had the full 15 points of height in Omnis 3, but conversion dropped them to only 12 points from the top of one line to the top of the next.

The most efficient way to properly align converted fields is to start at either the bottom or the top of the window and drag-select the first row, raise or lower those fields (as

appropriate) using the proper arrow key three times, shift-click to add the fields in the next row and repeat the process. It may be necessary to resize the window to accommodate all the fields on some crowded windows.

Button Region

The scroll bar that separated the fields portion of an Omnis 3 Plus screen from the button margin converts to a vertical line. I immediately remove this line since it doesn't fit into my design style and only seems to clutter the window.

The pushbuttons are converted to Omnis 5 pushbutton fields. They retain their labels and the Chicago font (standard for pushbuttons in any Macintosh application). If these buttons had been assigned sequences in the Omnis 3 Screen Information window, their procedures each call the appropriate procedure in the menu corresponding to the sequences from the original Omnis 3 Entry Layout.

Any window that did not have user defined buttons specified, but that retained the button margin, used the default pushbuttons. For any such screen, the entire set of buttons is converted to pushbutton fields, each with the appropriate default type. OK and Cancel button fields also appear on converted windows that had buttons in Omnis 3. They will need to be set to *Do not gray* to disappear when the window is not in *Enter data* mode.

Rectangles

Omnis 3 rectangles become real, moveable objects in Omnis 5. All Omnis 3 rectangles are converted to Omnis 5 rectangles with black borders of the proper width or pattern and white fill. They can be

resized or dragged to pixel precision or aligned to the current grid.

The converted rectangles will be stacked in the order they were created in Omnis 3. For example, if you had created a simulated shadow by drawing a double width line under a single width rectangle in Omnis 3, the line will show up in Omnis 5 *on top of* the rectangle (in stacking order) and will appear slightly shifted because of a slight difference in the definition of a line with a width of an even number of pixels. If you wish to retain this type of shadowing line, simply select the line and hit the down arrow key once to shift the line down one pixel on the window. (You could also select the rectangle and shift it *up* one pixel if that is easier.)

The only rectangle object that needs serious conversion is the one resulting from the conversion of an Array Window Rectangle. I'll cover that one next time when I discuss Array conversion.

Field Types

Omnis 3 Plus had three basic types of field for Entry Layout screens: Normal, Calculated, and Message. They each convert in predictable ways, but there are operational differences to consider. You may need to alter the conversion results to cover all your needs in the finished Omnis 5 application.

Message fields convert to Display fields with no field name. The message string remains intact in the Lookup table area and will display correctly as long as the string does not contain a slash (/) character. Field names referenced using semicolon notation will correctly convert to square bracket notation. Additional attributes (i.e., Invisible) convert as expected.

Calculated fields correctly convert to Display fields with the Calculated attribute checked. The calculation expression remains intact — unless the conversion required the addition of too many levels of parentheses as discussed above. Additional attributes (i.e., Zero shown empty) convert as expected.

Normal fields will convert to either Entry fields or Display fields, depending on the *Display only* setting in its Omnis 3 version.

But Entry and Display are only default modes assigned to fields in Omnis 5 — an enabled Display field is, in fact, an Entry field. Since the *Enable fields* command that results from the conversion of *Set field range* has this additional effect, it is important to check for procedure-field interactions after conversion. More on this next issue.

Field Attributes

Most of the attributes of Omnis 3 Plus Entry Layout fields translate directly to Omnis 5 window field attributes. *Autofind*, *Unique index*, *Upper case only*, *Negatives allowed*, and *Zero shown empty* all retain the same implications. *Invisible* fields can be made visible with the *Show fields* command, but are otherwise the same. Any field with the *Display only* attribute converts to a Display field.

Delete protected is no longer an attribute of fields in Omnis 5. This attribute is ignored on conversion. If your application needs this facility, a simple check in a window control procedure will do the same job — and can offer a more informative message to the operator.

Local fields used to automatically be *Display only* whether that attribute was also selected or not. I

didn't bother selecting *Display only* for these fields since it felt redundant to do so. In hindsight, this was not a good idea because *Local* fields can now be Entry fields and will be converted as such unless the *Display only* attribute was also selected in the Omnis 3 Library.

Defaults

Default values are no longer generated by the *Prepare for insert* command and fields no longer have a "Default" attribute in Omnis 5. Defaults are dropped on conversion (check calculations remain the same.) They must now be calculated and redrawn in a procedure.

If you chose to follow the guidelines I suggested in *Unlocking Omnis 3 Plus* for multi-user streamlined Insert sequences, you would already have done this. There we generally avoided using *Prepare for insert* in favor of *Prepare for insert with current values* and generated our defaults in the Insert sequence after clearing the Main File.

Pseudo Controls

There were many methods used by Omnis 3 programmers to simulate Macintosh radio buttons and check boxes. If the method explained in the Enter Data With Control chapter of "Unlocking Omnis 3 Plus" had been employed, the conversion to "real" radio buttons and check boxes is a relatively simple process. In this method, a single temporary variable was used to determine the state of a grouping of radio buttons. A message field was used to label each button and a temporary string field (usually a #Sx field) was used to display a selected or unselected button as appropriate using a special font.

continued on page 39

Data Entry Into Lists

Data entry involving Lists and List Fields seems to mystify many Omnis 5 programmers. There are many ways to accomplish this task. In this article, I will explain the two methods that I use most often.

The method that I use depends on how the operation is to take place. There are two basic possibilities that I have encountered so far. The first case is where I need to allow the entry of a number of “sub-records” at the same time as a Main File record is being entered (what I call a “secondary Main File” situation). Here the List is used as a temporary holding place for the record images that are to be written to the secondary Main File, but no records are written until the entire transaction has been entered and accepted.

The second case is where a new (individual) Main File record is to be added to a List of such records displayed on the current window. The second is the simplest to explain, so we'll start with that.

Insert Single Records

Consider the case where we have built a List of simple transactions for a customer and displayed them on a window in a List Field. We now wish to allow the operator to add another transaction for that customer to this List (and add a corresponding record to the File on disk) on this window. We may also want to allow modifications to existing records from the List one record at a time. This was the case for the Donations window accessed through the Customer window in the example disk for the last issue

(Vol I No. 2) of OmniScience. Figure 10 illustrates such a window.

The operations we want to perform on this kind of window are not very difficult to program and don't require much Omnis 5 magic or other trickery. The only requirement for field number order is that the List Field should not be within the range of the data entry fields for our new record entry. It can come either before or after that range. (Even this is not a strict requirement, but it helps the programmer conceptually. It also allows for redrawing of

Figure 10 Window for single record List entry

data entry fields without redrawing the List Field when necessary.) The data entry fields themselves have no special options selected except those required by the data (upper case only, for example). In fact, this window would be virtually the same if we were only adding a new record without adding it to a List — the only addition is the List field that shows the List contents.

The procedure for Inserting a new record would disable the List Field (in a reversible block) to avoid confusion for the operator. This simply means that, during the data

entry phase of the procedure, the List Field cannot become the current field. An alternative method would be to have a segment of the procedure associated with that List Field react to a #BEFORE event by setting the next action to a tab, a shift-tab, or a set current field as appropriate, but that method does not disallow clicking on the List Field. I prefer disabling the List field reversibly in this case, since we are entering records one line at a time and don't want to allow the operator to select another line while we're working on the new record.

Add to Bottom of List

The data entry to the Main File fields is done directly into the CRB as usual and the new record is saved to disk with an *Updatefiles* command. An *Add line to list* command line and a *Redraw windows* are then performed to place a record image for the new record into the List and display it on the window. If it is required to appear to insert the record image in some pre-specified sort order, a *Sort list* command should be performed between the *Add line to list* and *Redraw windows* commands (after guaranteeing that the proper sorts are still current).

It is usually desirable to make sure that the line in the List Field corresponding to the record displayed outside the List Field be highlighted. Also, it is good form to have no line highlighted while the new record is being entered (since it doesn't currently exist). The value of #L should be set to zero when the Main File is cleared and any defaults are calculated — *not* within a reversible block. After the record has been accepted and written to disk with *Update files*, the

Add line to list command will add an image of the new record to the bottom of the list. Calculating #L as #LN (the highest line number in the List) makes that line current so it will be highlighted by the Redraw windows command.

In the event that the operator cancels the operation, we would probably like to highlight whichever line had originally been selected. To do this, we would have to store the current value of #L in a temporary numeric field before calculating its value to 0. I usually do this in a reversible block, since I only need this value in the temporary during this procedure. In our Cancel Trap (the If flag false segment following the Enter data command), we can then calculate #L as its original value and relocate the records whose images are on that line before redrawing the window and quitting the procedure.

Here is a simple procedure that performs all the actions described above.

Add at Bottom

```
Begin reversible block
  Calculate INSERT as 1
  Set main file {Donation file}
  Set window control procedure
    240 {Add DO control proc}
  Set read/write files
    {Category file, Customer
    file, Donation file,
    System file}
  Calculate #1D0 as #L
  Disable fields 1 to 1
End reversible block
Calculate #L as 0
Clear main file
* Customer should remain, but we
  need to clear Category
Clear selected files {Category file}
* Set defaults in CRB
Calculate DO_DATE as #D
Redraw windows
Enter data
```

```
If flag false
  Ok message {}
  Calculate #L as #1
  Single file find on DO_SEQNO
    (Exact match) {1st(DO_SEQNO)}
  If DO_SEQNO
    Load connected records
      {Donation file}
  End if
  Redraw windows
  Quit procedure
End If
* Make sure System record in CRB
Single file find on SY_SEQNO
  (Exact match) {1}
Prepare for Insert w/ current vals
* Assign or modify values in or
  depending upon non-Main File
  records
Update files
Add line to list
Calculate #L as #LN
Redraw windows
```

Insert at Selected Line

If it is desired to insert the image of the new record at a location in the List field selected by the operator, a slightly different process would be used. After the operator had clicked on the line where the image is to be inserted (and then on the button to initiate the procedure), we would first want to clear all Files that have fields represented in the List field using the Clear selected files command.

For housekeeping purposes, I would do this even if I want to retain some of those records (Connected File records, for example) so that I can show a completely blank line in the List field during data entry. If a Connected File record is not represented in the List, it is not necessary to clear that File before inserting the List line.

We would then issue an *Insert line in list* command. Not specifying a line number will cause our blank

line to be inserted on the line that the operator had selected, moving the current contents of that line (#L), and all lower lines, down a notch. It is not necessary to specify #L as the line number for this command. If we needed to retain some of the non-Main File records that had existed on the originally selected line, we would have stored their RSNs in temporary numeric fields and then used a *Single file find* command to retrieve each of them. (In this example, it was not necessary to clear the Customer File since it isn't represented in the List. I therefore did not have to restore the Customer record to the CRB to properly make the new Donation record connected to that Customer.)

After calculating any other defaults, we would then issue a *Redraw windows* command and our List field would now show a blank line highlighted where our record image is to go if the data entry is accepted. We would not calculate #L as 0 in this case since we want to remain on the selected line.

After the new record is written to disk, we would then use the *Replace line in list* command to place its image on the line we had cleared for it. If for some reason the operator chose to cancel the procedure and not write a new record to disk, our Cancel Trap would have to include a *Delete line in list* command to do away with the blank line we had created. The line below (which had originally been on the current line) would return to its old position and be highlighted once again. We didn't have to worry about that in the procedure above because no new line was created until *after* the record had been written to disk.

Here is how our insert procedure would look modified for inserting a

record and placing its image at an operator-selected line of the List.

Add at Selected Line

```

Begin reversible block
  Calculate INSERT as 1
  Set main file {Donation file}
  Set window control procedure
    240 {Add DO control proc}
  Set read/write files
    {Category file, Customer
    file, Donation file,
    System file}
  Disable fields 1 to 1
End reversible block
Clear main file
* Customer should remain, but we
  need to clear Category
Clear selected files{Category file}
Insert line in list
* Set defaults in CRB
Calculate DO_DATE as #D
Redraw windows
Enter data
If flag false
  Ok message {}
  Delete line in list
  Single file find on DO_SEQNO
    (Exact match) {1st(DO_SEQNO)}
  If DO_SEQNO
    Load connected records
      {Donation file}
  End if
  Redraw windows
  Quit procedure
End If
* Make sure System record in CRB
Single file find on SY_SEQNO
  (Exact match) {1}
Prepare for Insert w/ current vals
* Assign or modify values in or
  depending upon non-Main File
  records
Update files
Replace line in list
Redraw windows

```

Insert in Sorted Order

What if our List field were already displaying record images in an order other than RSN order (the order in which the records were origi-

nally created)? Such would be the case if the List had been sorted after building or had been built on a Main File indexed field other than a Sequence type field. We may want our new record to be displayed in the proper order when it is added to the List.

We would perform the insert in the same way as in the first example, but we would have to do a little more work after adding the record to the List. The first extra thing we must do is to store the RSN for the new record in a temporary field so that we can locate its image in the List and highlight that line of the List field. (The name of a Sequence field for the Main File must be included in the definition of the List, but it doesn't have to be in the calculation of the List field.) Remember, the RSN is only available *after an Update files* command has been issued when we are inserting a new record. The calculation can be performed either before or after the line is added to the List, however. For efficiency, we can use the same temporary that we used for the restoral value of #L since we know that it is only a local value in this procedure (it was originally set in a reversible block) and we have past the point where we might have needed to restore the original #L value.

We would then *Clear sort fields* and *Set sort field* as required, unless we were certain that the Sort Field definitions were already appropriate. Next we would *Sort list* to get the record images in the proper order. At this point, we would have lost track of the image of our new record — but we prepared for this. We can now issue a *Set search as calculation* command specifying that the Sequence field equal the temporary numeric we used to store the RSN. Even better,

we can perform this in the reversible block and not disturb any Search Format we might otherwise have in force. We follow all this with a *Search list* command. Since the RSN is unique, we will *always* locate the right record. Finally, we would *Redraw windows* to highlight (and center, as necessary) the image of the new record. The Cancel Trap would be the same as in the first example.

We could have used a Search Format with a Calculation-type specification, but if we do this often enough, we'd end up with a lot of overly specialized Search Formats. We can't use a Comparison-type search criterion because the comparison is always done in the CRB, *not* in the List Buffer. Comparison searches cannot locate record images in a List in the current version of Omnis.

Here is how our insert procedure would look if the image of the new record had to be sorted into position.

Add in Sorted Order

```

Begin reversible block
  Calculate INSERT as 1
  Set main file {Donation file}
  Set window control procedure
    240 {Add DO control proc}
  Set read/write files
    {Category file, Customer
    file, Donation file,
    System file}
  Calculate #1D0 as #L
  Set search as calculation
    {#1=DO_SEQNO}
  Disable fields 1 to 1
End reversible block
Calculate #L as 0
Clear main file
* Customer should remain, but we
  need to clear Category
Clear selected files{Category file}
* Set defaults in CRB
Calculate DO_DATE as #D

```

```

Redraw windows
Enter data
If flag false
  Ok message {}
  Calculate #L as #1
  Single file find on DO_SEQNO
    (Exact match) {1st(DO_SEQNO)}
  If DO_SEQNO
    Load connected records
      {Donation file}
  End if
  Redraw windows
  Quit procedure
End If
* Make sure System record in CRB
Single file find on SY_SEQNO
  (Exact match) {1}
Prepare for Insert w/ current vals
* Assign or modify values in or
  depending upon non-Main File
  records
Update files
Calculate #1D0 as DO_SEQNO
Add line to list
Clear sort fields
Set sort field {as you wish}
Sort list
Search list
Redraw windows

```

Locate Record for Edit

Beyond these methods for inserting new records, it is usually also desirable to be able to make changes to a selected record on a window such as I have described here. To be able to do this, it is necessary to have a procedure for the List field that locates the record or records associated with the selected line when the operator clicks on the List field or uses List navigation techniques.

Here is a simple field procedure for a List field that performs that task. Remember that the #CLICK message is sent whenever a new line in the List field is selected by any means (see Vol I No 1, p. 21), but it is *not* sent if the operator clicks on the currently selected line.

```

If #CLICK
  Single file find on DO_SEQNO
    (Exact match) {1st(DO_SEQNO)}
  Load connected records
    {Donation file}
  Redraw named fields DO_NUMBER
    to DO_NOTES
  Redraw named fields CA_CODE
    to CA_DESCRIPTION
End If

```

The record can then be edited in the same way it would normally be edited. You will want to disable the List field in a reversible block in your edit procedure to prevent the operator from accidentally retrieving a different Main File record while in the *Prepare for edit* mode.

Secondary Main File

There are many occasions when we would like to be able to load an entire transaction involving multiple records for a number of Files into the computer all at once and then either commit them to disk or cancel from the entire process. Most multiple File database programmers must go to lengthy and complex procedures, often involving many windows and many command keystrokes, to perform this task — and the programming platforms they use must include complex and laborious (and sometimes not very reliable) commands that are usually named COMMIT and ROLL-BACK to allow the programmers this capability.

Lists and List Fields in Omnis 5 make this job a lot easier. With proper (and not overly complicated) programming techniques using these features, we can create single windows that allow the operator to input very complex transactions with only a single command keystroke. They commit the transaction by clicking the OK button, at which point we can also have a

control procedure check the data for validity before writing it to disk. There is never a partially completed transaction to roll back if the operator decides to cancel the process. Although no computer system can claim to be 100% bulletproof, the chances of only part of the records of a transaction being written to disk in the event of a crash is *extremely* minimal when using the techniques listed below.

On the example disk for the first issue of OmniScience, I included a window for inserting new invoice transactions to allow the user to further test the reporting technique of using a List as a Sort Buffer. It was not my intention to provide detailed explanations of that window at that time — I only intended to offer it as a tool for inserting test records. However a great deal of interest was shown in learning how to build such windows, so I will use that as the example for this part of this article. A simplified version of that window is shown in Figure 11.

On this window, both an Invoice Header record (connected to a Customer record) and one or a number of Line Item records (connected to both the current Invoice Header and to their respective Product records) are inserted as a group. The Invoice Header File is the Main File during the data entry, as the record in that File has to be written to disk before any Line Item records for that invoice. After the Invoice Header is written to disk, the procedure changes the Main File setting to Line Items and creates a record for each record image captured in the List Field during data entry.

The key element that allows for speed and simplicity of data entry to the List Field portion of this window is the Local attribute as-

signed to the entry fields located just above (and coming in field number order just after) the List Field. A review of the Local attribute will help the rest of this explanation along...

The Local Attribute

If a field is marked as "Local", it can be said to "belong" to the most immediately preceding non-local field in field number order on that window. There are a number of useful and intriguing implications to this statement.

One result of this attribute is that the Local field is reevaluated and redrawn when the cursor leaves the field to which it is Local. We might, for example, have a display field whose calculated value depends on a value from a data entry field. Making the display field Local to the entry field would cause the display field to be updated each time a new value is left in the entry field. Ways of leaving the entry field include tabbing, shift-tabbing, clicking on another field, and selecting OK or Cancel.

Another result of the Local attribute is that if calculations performed in a Local field affect the value of its "lead" field (the most immediately preceding non-local field), the lead field is also redrawn with its new value. We can say that the Local attribute is "reflexive", to use a term borrowed from linguistics. That is, it "reflects" back to the lead field.

It is possible to have a chain of Local fields. Each is Local to the lead field of the chain and to each Local field that precedes it in field

number order. Local fields can be display, entry, radio button, or checkbox fields in Omnis 5. Changes to any of these will reflect back to the lead field.

Fields that are Local to a List field and are included in the definition of the corresponding List have an even more intriguing attribute. They are actually windows on the values in the List Buffer rather than in the CRB. So a data entry field that is Local to a List Field (and whose field name is in the definition of the corresponding List)

Figure 11 Window for multiple record List entry

is actually a data entry window directly into the List Buffer for the corresponding List, *bypassing the Current Record Buffer*. Furthermore, if that field's value is used in the calculation of the display string for the List Field, the current line of the List Field is *automatically* redrawn to show the result of the new value when the cursor leaves that entry field due to the reflexive nature of the Local attribute.

Although a List Field can only display values from a List, it is still possible to perform data entry directly to a List using this attribute. There are two basic problems that we have to surmount to effectively use this technique. The first is that

there is no way that we can replace individual values on a line of a List through calculations in procedures in the current version of Omnis. The other is that we have no automatic way of tabbing between lines of the List during data entry.

List Buffer vs. CRB

It is quite possible that we might need to modify some values in our List through the use of procedure commands. We need to be cautious when doing this to not lose any values already in the current line of the List. Here's why...

The only way we can put a changed value into a field in a List (in the current version of Omnis) is to use the *Replace line in list* command. This command replaces the *entire* line, not just a single value. Since our data entry fields Local to the List Field bypass the CRB, we must first *Load line from list* to make sure that the CRB accurately reflects what is in the current line. Then we can perform our calculations and *Replace line in list*, knowing that we are replacing unchanged fields with their original values.

Another implication of doing data entry directly into the List Buffer is that *we need to be on an existing line of the List* to successfully enter data. If our Insert procedure has cleared the List, no lines currently exist and the list pointer (#L) is set to line zero. We need to both *Add line to list* and set #L to 1 before this process will work properly.

Processing Order

It is important to remember in what order things happen when using

field procedures in Omnis 5. In general, field procedures for data entry fields (or any other fields) only begin when all the implications of that fields attributes have been carried out. For example, if a data entry field has the automatic find attribute selected, the automatic find will be performed *before* a field procedure segment triggered by the #AFTER message flag. If that field is Local to a List Field, any List Field values affected by the automatic find will be reflected in the List Field automatically, but any changes to List Field values caused by that fields procedure will *not* be reflected without redrawing either the window or the List.

If on our entry window we perform an automatic find on the Product Code field, the Product Description and the Product Taxable field values (both in the Local entry fields and in the List Field) will be automatically drawn. If our field procedure then calculates a default Item Unit Price value based on the Product Selling Price, that value must be specifically redrawn to both its entry field and to the List Field. Redrawing the entry field alone will not redraw the List Field — the phase during which the Local attribute causes this to happen automatically has long since past.

Tabbing to Next Line

Navigating different List Field lines by tabbing among its Local entry fields is also not automatic — but we would like it to *seem* automatic to the operator. We can use event management techniques to do this. There are only a few possibilities that we have to deal with.

On tabbing forward, when we reach the last field in field number order on a given line, we would like to tab to the first field on the next line —

unless we are already on the highest line number we want to allow (only 30 lines on an invoice, for example). In that special case, we would like to tab to whatever field comes after the List Field entry group. In our example, that would be the Payment Details field.

On tabbing backward, when we reach the first field in field number order on a given line, we would like to tab to the last field on the previous line — unless we are already on the first line. In that case, we would like to tab to whatever field comes before the List Field entry group. In this example, that would be the Customer ID field.

Field Procedures

Taking these concepts into account, we can now examine the procedures in certain key fields. The Inventory Code field, for example, performs a number of important tasks. It locates an Inventory record through its *Automatic find* attribute and reads it into the List buffer (it is *Local* to the List field). If we tab forward and leave this field blank, we can use this as a signal to tab out of the List entry area and finish the lower part of the window. If we tab forward and an Inventory record has been located, the field procedure can transfer the Selling Price value to the Line Item Unit Price field.

Tabbing backward presents us with two choices. If we are already on the first line of the List, we should have the cursor leave the List entry area and go to the Customer ID field. If we are on any line other than the first, however, we would like to move up a line and place the cursor in the last entry field on that line (the Unit Price field). The following field procedure performs all these tasks.

```
INCODE Field
If #TAB
  Load from list
  If not(IN_SEQNO)
    SNA set current field
      {[nam(IV_PAY_DETAILS)]}
  Else If not(IT_UNITPRICE)
    Calculate IT_UNITPRICE as
      IN_SELL_PRICE
    Replace line in list
    Redraw windows
    SNA set current field
      {[nam(IT_QUANTITY)]}
  End If
Else If #STAB
  If #L=1
    SNA set current field
      {[nam(CU_ID_NUMBER)]}
  Else
    Calculate #L as #L-1
    SNA set current field
      {[nam(IT_UNITPRICE)]}
    Redraw windows
  End If
End If
```

The other key field in this data entry process is the Item Unit Price field. This is the last data entry field in the List entry area. The procedure for this field is used to calculate and display the latest totals figures for the invoice and to add new lines to the List if necessary. Tabbing backward from this field has no special consequences, but tabbing forward will cause us to leave this line of the List. There are three possibilities as to how this could proceed.

If we have already reached the limit of lines allowed in the List, we would like to leave the List entry area and place the cursor in the Payment Details field. If we are currently on the highest line number created so far, we need to create another blank line, move the pointer to that line, and move the cursor to the first field in the List entry area (Inventory Code). If we

continued on page 40

Tips & Techniques

This section is used to describe short, useful features or techniques that can make your Omnis programming easier or more efficient. Sometimes the specific example may not be of direct benefit to your immediate projects, but the methods employed will be helpful.

Age in Years

Many applications need to have the facility to determine a person's age on a specified date. Medical applications and school applications are the two most likely to have this requirement, but there could be countless others.

I have seen a number of algorithms proposed for determining a person's age in years on a specific date, but none have been 100% accurate. The question came up in the Streamlining for Performance class I gave in Washington, DC in April, and we came up with a perfect solution. The core of the solution was offered by Susan Frey of Orion Computer Systems who attended the class.

The solution is really very simple. Consider what you do to figure out how old a person is. You first determine the difference between their birth year and the year of the date in question. This gives you a close estimate. Then you determine whether their birthday has passed for that year. If it hasn't, you subtract a year from the original estimate. It works every time!

So how do we do this in Omnis 5? The key element that Susan proposed is that we assign what I call a pseudo-Julian date value to each date to determine their relative positions within their own year.

This is done by multiplying the month number of the date by 100 and then adding the day number. July 15th would have a value of 715, for example. February 29th (229) would always be less than March 1st (301), so leap years wouldn't cause us a problem. All we had to do was construct an expression that incorporates all these concepts. Here is our result.

```
dtcy(#D)-dtcy(dat(BIRTHDATE))-
  (dtm(#D)*100+dtd(#D)>
  dtm(BIRTHDATE)*100+
  dtd(BIRTHDATE))
```

The first two parts of the expression determine the basic year difference. We used the combined century and year value to be able to span century breaks. The longest subexpression calculates the pseudo-Julian values for the current date and the birthdate and then compares those values.

Notice the use of precedence to avoid overusing parentheses—the comparison is performed last because all the other operators take precedence over it. If the person's birthday has not yet occurred in this year, the pseudo-Julian for the current date is greater than that for the birthdate and the comparison is true. This causes the expression to subtract a year from the original estimate. If the birthdate pseudo-Julian is greater than *or equal to* that of the current date, the subexpression evaluates to zero and the original estimate stands.

Remember that if you want to “hard code” a date value by typing the date string into the expression, you will also have to use the date conversion function *dat()* to convert that string to an actual date value.

Expanding Boxes

Many people like to spruce up their reports with graphic elements to make them more presentable or more readable. Unfortunately, we don't have any graphic elements in Omnis 5 reports that expand with the data being printed. All graphic elements created by Omnis or pasted into Omnis have a fixed size when printed (although we can stretch them horizontally and/or vertically when placing them on the report template). With a little maneuvering, though, we can use some fixed-size elements to build an expanding box.

Let's suppose that we would like to have a box surround all the lines in a subtotal grouping of record sections. That is, we would like a border around the entire group, not around each record image.

The key concept here is that each graphic element belongs to the row of the report *in which its top-most row of pixels lies*. The element will be printed in its entirety *even if it crosses over into another section*. Here is how we use this information to solve our problem.

We will build our box like a telescoping water cup. The top of the box will be in the Subtotal Heading Section and will consist of a horizontal line and two vertical line segments. The vertical segments will form corners at the ends of the horizontal line and will be long enough to extend slightly into the Record Section banner. We can't see exactly how far they extend because the section banner is printed to the screen *after* (and, therefore, on top of) all other elements. In printing, the vertical segments from the Subtotal Heading will extend into the first line of the first Record Section.

The bottom of the box will be another horizontal line. This one will be in the Subtotal Section following the Record Section. There will be no vertical line segments in this section, since we can't make a graphics object extend *above* the line it resides on.

The telescoping sides of our box will be made up of vertical line

jects that span sections on the Report Template. Specifically, it's hard to line up the bottom corners vertically so that the vertical segments from the Record Section *just meet* the ends of the horizontal line in the Subtotal Section.

Fortunately, we can "turn off" the section banners with the *Show narrow sections* command from

Sort Field #4 changes and the box should come to proper corners if you have performed the setup operation outlined here.

In general, the subtotal level that should be used for the bottom of the group is the highest numbered subtotal level declared in the Report Format. It is possible to have boxes surround different levels of subtotal groupings, but the method is a little trickier.

Suppose we wanted a box with a different weight to surround the Subtotal Level #3 group. A line of the

proper weight would be placed on the line above the one we already have in the Subtotal Heading Section along with matching verticals. An invisible field for a temporary numeric variable would also be placed on that line and given the *No line if empty* attribute. The value for that variable would be calculated in a separate field in the

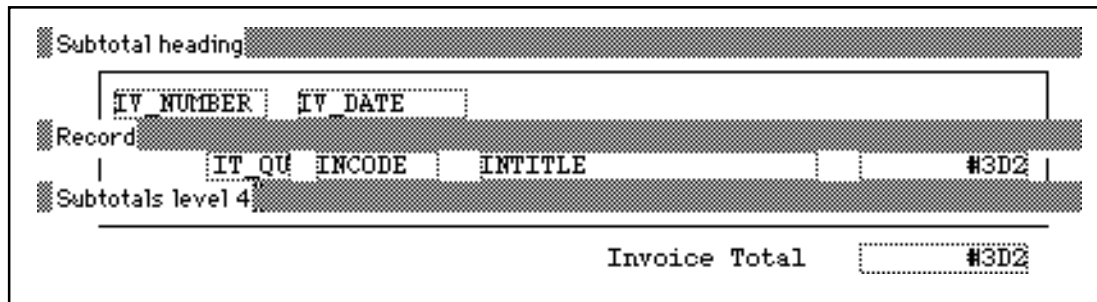


Figure 12 Subtotal Heading, Record, and Subtotal Sections as they normally appear. Notice the effect on the vertical line segment placement.

segments placed in the Record Section. These will start just below the topmost pixel of the top line of the section and will extend into the Subtotal Section banner that follows the Record Section. They must line up *precisely* with the vertical line segments in the Subtotal Heading Section to act as extensions of those segments. Figure 12 shows what these sections will look like when the line segments have been properly placed.

Proper alignment of the bottom corners

would be a trial-and-error process if it weren't for another feature of the Report Modification mode. You see, even though the section banners don't print, they take up visual space when we are laying out our reports and their image interferes with precise placement of ob-

jects in the Modify menu (command-N). This shows the section banners as dotted lines with no effective height instead of as gray bars as in Figure 12. Notice that the section banner images lie on the topmost row of pixels in the top line of each section. This can be seen most clearly in the Record Section in Figure 13 where the dotted section line blends

with the top outline of the field images in that section.

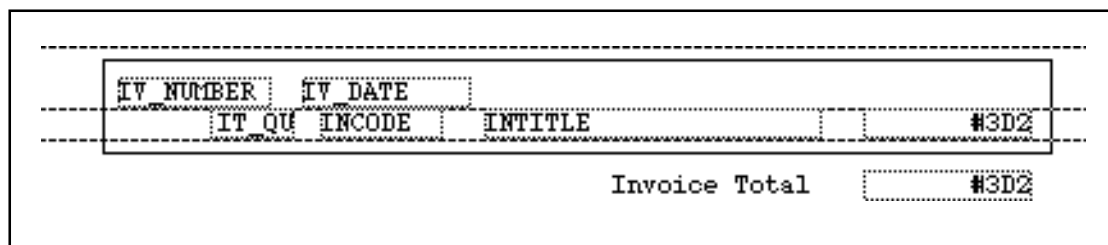


Figure 13 Subtotal Heading, Record, and Subtotal Sections with Narrow Sections turned on. Notice how the vertical segments are now aligned as they will print.

with the top outline of the field images in that section.

When the report is printed, another line segment is printed on the left and the right sides for each Record Section. The group is completely enclosed when the value of

corresponding subtotal section (where it would be turned "on"), and again in the Subtotal Heading Section where it would be turned "off". The verticals in the Record Section would have to be long enough to span Subtotal Level #4 and the Subtotal Heading.

Haven

Continued from page 1

Certainly there were questions to be answered and points to be clarified, but it was very obvious that developers were the center of attention not just at the conference, but in the daily activities of the Omnis distributor. And both the developers and Solutions seemed to be reaping tremendous benefits from this cooperation!

The first level of support came in recognizing that the developers and the distributor have separate but equal roles to play in spreading the use of Omnis 5 nationwide. The distributor makes sure that there are plenty of Omnis packages available for sale, does the bulk of the advertising and publicity, and provides efficient and excellent technical support to those requiring it. They also generate and provide many leads to those devel-

opers who are capable of handling them. The developers create either custom or off-the-shelf database applications using Omnis 5 and introduce the product to their expanding client bases. In-house corporate and government developers, as well as independents, provide success stories and influential references to aid in the publicity battle.

The result is that Omnis 5 far outsells both 4th Dimension and FoxBASE in that market, on the Macintosh side, and is making tremendous gains in the Windows arena. In fact, I hear that Solutions sells many more Omnis 5 packages than does Blyth US (or did in those recent accounting periods) — with only 10% of the US's potential market!

The technical support provided by John Witstyn and his staff is also first rate. The telephone is gener-

ally answered by a human being who intelligently directs each call to the appropriate staff member. If an answer isn't immediately available, it is determined as quickly as possible, either by diligent in-house research, calling the UK, or (on one or two rare occasions) by calling my office. The client (developer) is called back as soon as possible with the answer. There is no ignoring of requests, there is no indicating that the caller doesn't have sufficient intelligence to use the product, and there is usually no significant delay in receiving an answer that works.

I was told that (at least in early 1991) an average of four corporate leads was received by Solutions *every business day* from Microsoft alone. These leads were all distributed among developers with the proper background and expertise, generally in the proper geographic area to be of optimum assistance.

Reviews

Continued from page 1

The person who did the benchmark testing was also an interesting case. He configured the "multi-user" tests to be a single workstation accessing a database on a server machine in multi-user mode. Since I insisted on more than one File being involved, a two-File system (Customers and Bills) was used. He told me with pride that this was a subset of a three-File system he had designed for a medical office and that the test configuration was typical of the real world because only one person (the receptionist) ever actually does data entry anyway — the doctors in the back room only look up things on occasion.

My insistence on a six-File system (Invoicing structure with Constants and Daily Sales Tally Files

to produce possible record locking conflicts) and five workstations attempting to simultaneously perform the same tasks was labeled "unrealistic" and "overly complex". Both the other reviewer and the tester agreed that "no one would ever try to do *that* with a *Mac* — they'd get a *minicomputer* if they needed *that* much work done!"

Imagine your favorite sportscar being reviewed by a person who had only ever successfully driven a horse and buggy and who didn't quite believe that the sportscar actually worked. The first thing that would happen is that the reviewer would insist on hitching a horse to the front end. In the article, he would then proceed to complain about everything from the apparent slowness of the vehicle to his inability to get a good view of the road! These things would naturally

show up on the test results because the benchmarks would be biased toward horse-and-buggy issues rather than testing the vehicles ability to move people and luggage quickly, efficiently, and safely.

The point here is *not* to point fingers at the stupidity or inexperience of such reviewers (although some of the anecdotes are very amusing now that enough time has passed), but to express concern about the quality of information available on which people base their software and consulting purchase decisions.

The world of information management is not a simple one, and people who approach it as being potentially simple do not reap the benefits that could be theirs with a deeper understanding. This applies to programmers and consultants as well as to reviewers...

Their greatest problem was in finding or developing enough qualified developers to handle the load! Solutions specifically did not want to take on development projects, since they felt there was enough to do selling the product and supporting developers.

By the way, Solutions is actively seeking new people with expertise developing applications in Omnis 5. Australia welcomes immigrants with technical know-how and it seems like a pleasant place to live. A number of developers at the conference also indicated an interest in working with developers in the United States and Canada (or, I suppose, anywhere else they may be) who have applications that might be saleable in Australia (perhaps with modifications to handle local vocabulary and accounting customs, etc.). I would be happy to help bring any interested parties together.

Conversion

Continued from page 29

The final manual conversion of this arrangement is easy. First, we convert each of the former message fields to a radio button field and assign it the field name of the variable used for the group. Then we transfer the string from each labeling message field into the label area of our radio button fields and delete the old labeling fields.

Check boxes use separate temporary numeric variables. Again, a message field was used as a label for each and a #Sx field was used to display the selected or unselected check box character using a special font. The conversion is performed the same way as that for radio buttons, converting the message field that displayed the check box into a true check box field.

Example Disk Program

Even the greatest verbal explanation is worth a lot more if accompanied by working examples. It is my intention to make such examples available with each issue. I recognize that not everyone will feel the need for these, so I have made them an option.

Each issue will contain at least one disk offer and often two or three. These will be at different levels of complexity or commercial value and will be priced accordingly.

For example, each issue will at least offer a disk containing the examples from the Tips and Techniques section and other articles of that issue, as well as variations and extensions of those examples. The records contained in the associated data file(s) may also hold useful information. This will be an inexpensive disk, usually in the \$20.00 to \$30.00 range.

If there are more complex principles in some article that require a more comprehensive application for demonstration, a disk with such an application will be offered in the \$50.00 to \$100.00 range.

Finally, I have been working for over three years on a project code named the "Database Construction Set" for which I have received many requests. The project just got too big and there seemed to be no good pricing and licensing scheme for it. I have decided to release it in a "serialized" form (as in "tune in next time for the next exciting..."). Most issues will have one or a number of articles about a specific design topic that relates to one of the modules of these "Solution Packs". For example, this issue will have a lengthy discussion on computer-

ized invoicing systems. Invoicing modules that you can patch into your applications *with no additional royalties* are currently being developed. A licensing agreement will accompany each Solution Pack basically stating that you will retain my copyright in each format, include the phrase "portions copyright by David Swain" on your About window and in any manuals, and not distribute the module as open code (or share it among your friends). The Solution Pack modules will generally fall in the range of \$200.00 to \$500.00.

I hope these disks will be of use to many of you and that the extra time I spend preparing them for you will be appreciated.

This issue's disk contains an application demonstrating:

Multiple Main File Reports

Data Entry to Lists

Flat File Image of Complex Structure

Streamlined Algorithms

Age of Date in Years

and many other related techniques.

Because this month's application is so much more complex than those previously offered, this disk is priced at \$30.00.

Also available this issue

Omnis 5 Shell Application \$100.00
A great starting point for any application. It will save you hours on each new project!

General Ledger Solution Pack \$200.00
The tools you need to add bookkeeping to your applications. A royalty-free basic General Ledger application with commented code.

List Entry

Continued from page 35

are on any other line of the List, we simply want to move the pointer down a line and move the cursor to the first field. Here is a procedure that does all these things.

```
ITPRICE field
If #AFTER
  Call procedure Invoice Entry/229
  {calc totals}
If #TAB
  Clear selected files
  {Inventory file,Line Items}
If #L=#LM
  SNA set current field
  {[nam(IV_MESSAGE)]}
  Quit to enter data
Else If #L=#LN
  Add line to list
  Calculate #L as #LN
Else
  Calculate #L as #L+1
```

```
End If
SNA set current field
  {[nam(IN_CODE)]}
Redraw windows
End If
End If
```

It is important to bear in mind that no records have been written during this data entry phase. What we have done is to enter record *images* that we then wish to *convert* to actual records stored on disk. Our Insert procedure would first create the Invoice record, then set the Main File to Line Items and process each line in the List to create a Line Items record connected to the Invoice record that was just created and to the Product record whose pointer was stored on that line. This processing is done very quickly, and the chance of a system failure or power outage during the process is extremely small.



OmniScience is published bi-monthly by David Swain, 1418 Park Avenue, Alameda, CA 94501. Address all editorial correspondence, requests for special permission, subscriptions, or requests for bulk orders to The Editor, OmniScience, 1418 Park Avenue, Alameda, CA 94501.

Domestic subscriptions (in US funds): 6 issues, \$195.00. Canadian subscriptions, \$205.00. Outside the USA and Canada, \$220.00. Single issues are available for \$40.00 domestic, \$42.00 Canada, and \$45.00 elsewhere. Send subscriptions, changes of address, and fulfillment questions to OmniScience, 1418 Park Avenue, Alameda, CA 94501.

Copyright © 1990, David Swain. All rights reserved. No part of this journal may be used or reproduced in any fashion (except in brief quotations used in critical articles and reviews) without the prior written consent of David Swain.

Publisher:	David Swain	Production:	David Swain
Editor:	David Swain		Jeanette Colgan
Design:	David Swain		
	Gwenn Connolly		

OmniScience and Polymath are trademarks of David Swain. Omnis and Omnis 5 are trademarks of Blyth Software, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. 4th Dimension is a trademark of ACIUS, Inc. and ACI. FoxBASE is a trademark of Fox Software, Inc. dBASE is a registered trademark of Ashton-Tate Corporation. Macintosh, ImageWriter, and LaserWriter are trademarks of Apple Computer, Inc. IBM PC and IBM PS/2 are trademarks of International Business Machines.

In the Next Issue

Volume I Number 4 extends our Design discussion to Accounts Receivable and expands on conversion techniques from Omnis 3 Plus. I have received many requests for both of these items. Articles in this issue will include:

Receivables and Payments

Not all sales transactions are complete on printing the invoice. Often it is still necessary to collect the money at some future date. A (continuing) short course in accounting.

Computerizing A/R

Accounts receivable systems allow us to explore a new data structure that has a number of other uses. We will develop a design that allows for flexible tracking of multiple and partial payments against receivables.

Ungraceful Exits

A number of Omnis programming techniques in common use today can lead to long-term memory management bombs. This article shows you how to avoid these "time bomb" techniques while avoiding the dreaded Clear procedure stack command.

Converting from 3 to 5, Part 2

By popular demand, this article outlines more problems you will face in converting existing Omnis 3 Plus applications to Omnis 5. It also gives some useful tips and techniques.

List Manipulations

Here are two more important techniques you can use with Lists. One technique finds all the common elements in two Lists (the intersection of the elements), the other performs a true merge between two Lists (the union of the elements in both without duplication).

Tips and Techniques

More clever algorithms and Omnis 5 hacking for your viewing pleasure. Requests and suggestions are welcome and will be properly credited.