

Event Management Revisited 2
Magic is often more fascinating when you know how the trick was done. Event management isn't magic, but when you understand how it works (and how to make it work for you), you'll have a more fulfilling experience.

New Features of Lists 8
And you thought that Lists were powerful in Omnis 5! Here is an introduction to some impressive new features and how to use them.

Adventures with the Debugger 14
At last! No more OK messages — and a chance to really see what's going on when OMNIS executes a procedure.

eval() Functions 20
A slick new feature to convert a string into an actual expression and evaluate it. But that's not all — we also have another opportunity to explore how OMNIS 7 operates internally.

jst() Techniques 24
Here are some tips on getting more use from one of my favorite functions.

Ad Hoc Reports 26
Now we can give our users the ability to craft their own reports — to a certain extent.

Tips and Techniques 32
Angular format conversions, benign hacking, and much more...

About the Author 36
The folks at Blyth suggested that you might want to know a little bit about me and why you are receiving this newsletter.

Class Schedule 37
If you need training on OMNIS 7, help is on the way. The longest-running OMNIS-based course offering has a new nationwide schedule.

Example Disk Program 38
And another description, etc.

Products 38
A growing list of OMNIS-based example disks, auxiliary software and publications.

Fax Survey 39
I can serve your needs best if I know what they are. This is your chance to give me some feedback and make suggestions. Mail is fine if you don't have a fax machine.

OMNIS – The Next Generation

by David Swain

Progress sometimes occurs very rapidly in the world of computer software. It seems like only yesterday that I started delving into the secrets of Omnis 5 and here we are, two and a half years later, facing the challenges of a new generation of even more complex database software. Time flies when you're having fun, I guess.

Let's get one thing clear from the beginning — there is nothing “simple” about creating and maintaining transactional database applications. The word “database” is used to represent many different forms of information. The most simplistic of these views considers a database to be just a lookup table of stored information (see any MacUser article by Jean-Louis Gassée that mentions databases). There is much more to the subject than you commonly see addressed in the computer media.

Within these pages we will be dealing with systems for modeling the information flow of an enterprise (or a portion of it), often tracking that information as it comes into existence. The texture of the information in these systems can often become quite intricate and complex as we allow for more flexible access to that information for updates and reporting.

continued on page 40

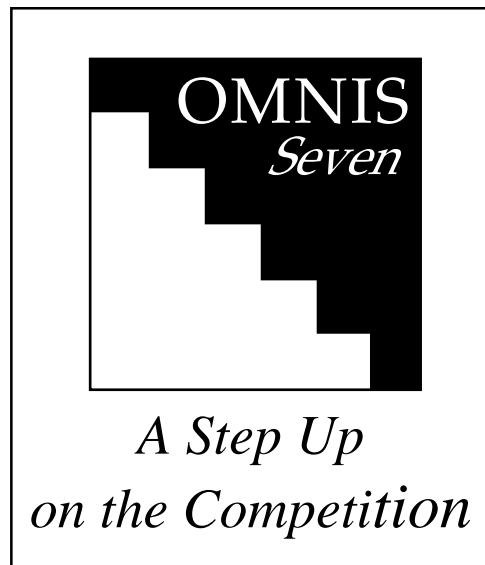


Figure 1 The new OMNIS 7 logo — with my own caption added for good measure.

Welcome to OmniScience Volume II

Welcome to the world of OmniScience. I hope you derive great benefit from the information I provide here. There are a lot of exciting new features to explore with you — and new uses for the old features are being developed every day! As OMNIS evolves, I intend to keep you up to date.

For those of you who are subscribers to OmniScience Volume I, OmniScience Volume II is an entirely different product in spite of the similar appearance. Volume I is very broad in scope and includes a great many discussions on database design — both generic and OMNIS-based. Volume II is strictly in-depth technical information and is essentially offered as a supplement to tech notes from Blyth.

continued on page 36

Event Management Revisited

In Volume I Issue Number 1 of OmniScience (known as *The Basics*) I introduced the fundamentals of event management. Unfortunately, not many people have had a chance to learn from that article — and I have seen a lot of code and fielded a lot of questions since then from students and clients that indicate there are some misconceptions that still need clearing up. So I will dig a little deeper into the mysteries of control procedures and message variables here.

Control is Sequential

First of all, all programming is basically sequential in nature. It may appear that some processes are happening in parallel (or at the same time), but this is not the case. For example, when a subroutine is called using the *Callprocedure* command, processing is shifted to that subroutine until it has finished or is exited (quit). Processing then returns to the calling procedure and continues.

In a similar way, event management procedures are also processed in sequential fashion. The *Enter data* command causes the current procedure to *pause* while the operator is allowed input of some kind to the process (typing, tabbing, clicking, etc.). During this time, the various field attributes and control procedures react to this input *in their turn*. Keep in mind that the procedure that invoked the *Enter data* mode is still running — it is just “on hold” until the OK or Cancel button (or the keyboard equivalent) is ex-

ecuted by the operator, signalling the end of the *Enter data* phase.

Field attributes (automatic find, local, etc.) take precedence over all control procedures with regard to their order of execution — and they have their own processing hierarchy as well. Then the field, window, and application control procedures come into play to deal with the message stream initiated by the operator’s action. Field, window, and application control procedures are executed *in that order* — and *only* after all field attribute programming steps have been com-

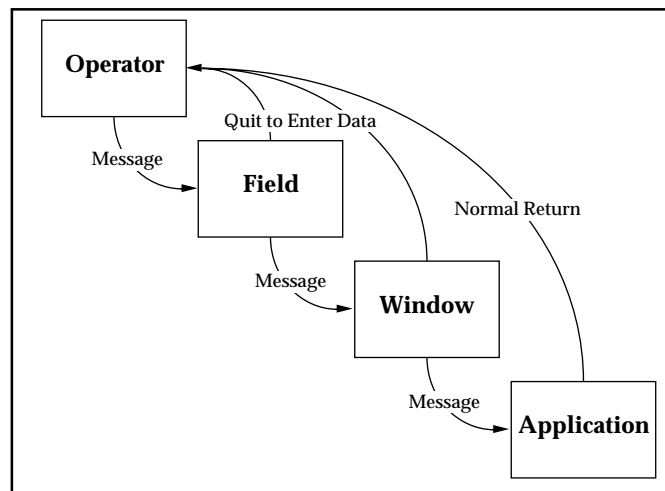


Figure 2 The flow of control in OMNIS 7 event management.

pleted. Understanding the flow of this process will help you to write more efficient applications.

Field Attributes

The “screen program” that I wrote about in my early books on Omnis 3 Plus is still very much alive in OMNIS 7. Window fields are object-oriented programming constructs that can perform significant work. The procedures that they carry out are hidden from our

view to simplify our work, but a lot of C++ coding by those clever folks at Blyth has gone into making them operate as they do.

Most of the field attributes, such as upper case conversion or zero shown empty, perform functions that only affect the current field. The *Automatic find* attribute affects the CRB (by reading in a different record), but doesn’t affect the display of values in other window fields. The *Local* attribute, on the other hand, can affect the values displayed in a number of fields at once. This deserves a closer look.

Local Field Chains

If a field is given the *Local* attribute, it is re-evaluated and redrawn whenever the preceding field (in field number order) or the most immediately preceding non-local field is acted upon. If that field (the one acted upon) is a data entry field, the local fields immediately following it will be redrawn whenever the cursor leaves it — whether the operator tabs, shift-tabs, clicks on another field, or clicks on the OK button. If that field is a radio button, a check box, or a list field, etc., the redraw of local fields is triggered by a click or double-click on that “lead” field.

The local fields can be used to perform work, such as calculating new values for variables or even locating records by the use of the *Automatic find* feature. True, these operations can also be performed by field procedures, but these simple operations are more efficiently done — and done sooner — by the field attributes mentioned here.

The Message Stream

When the operator performs some action, a collection of messages is sent into the message stream. The values included in this collection are the names of the message variables that have been “turned on” by the operator’s action and the name and number of the current field. In the special case of a window being brought to the top, the messages are accompanied by the name of the Window Format and the number of the procedure that is called (procedure 0). These messages can be viewed at any time by examining the value of *sys(84)*.

The messages in the message stream remain the same for a given event cycle — *even if you have used a SNA command*. The SNA command only redirects the outcome of the event cycle — it doesn’t affect the original messages. If this were not the case, we couldn’t have a *SNA perform default action* command because the default actions would be lost and forgotten.

This also means that *all the messages apply to all the levels of control in a given event cycle*. A #BEFORE or #AFTER event is detectable not only at the field level, but at the window and application control levels as well — though it is most likely to be useful at the field level. We’ll talk more on this later.

Events and Attributes

The event messages that are sent to the control procedures are not completely formed until field attribute needs are fulfilled. Prove it to yourself by placing a local calcu-

lated display field after an entry field and make the calculation for that field simply *sys(84)*. (See Figure 3.) The message string you will see when you tab out of that field will be little more than #EDATA. If you had entered that field by shift-tabbing, the #SHIFT message would seem to persist, but that is about all. You will notice that the field value only changes as you leave the entry field. This is because the local attribute only operates under that condition.

Just so you know, the debugger is not designed to examine this effect. It is intended to show us a trace of the *procedure commands*

field since the field procedure is now controlling the redraw.

```
If #AFTER
  Redraw numbered field
    display field number
End if
```

All this does is force the calculation to be performed *after* the control procedures begin their work. You will notice that the message string is much longer. Not only do we see #EDATA, but we see the #AFTER event, the method of leaving the field (#TAB or #STAB), and the field name and number of the field the cursor is leaving.

By the way, the cursor has not quite left the field at this point. It *intends to* — and will if events are not redirected — but it is the *intent* to leave the field that these messages are communicating to us. The control procedures we set up can then intercept the intended action *before it is carried out* to perform conditional tests and calculations and to determine whether the intended action should be allowed or redirected.

If we click OK or Cancel while the cursor is in the entry field, those event strings will be displayed as well, because both OK and Cancel generate an #AFTER event for the current field. We didn’t see a change to the displayed message value in the display field on an OK or Cancel when we were relying on the local redraw, because those events hadn’t quite happened yet. Clicking OK after modifying the field value would show the #MODIFIED message. Generating an OK by pressing either the Return or the Enter key

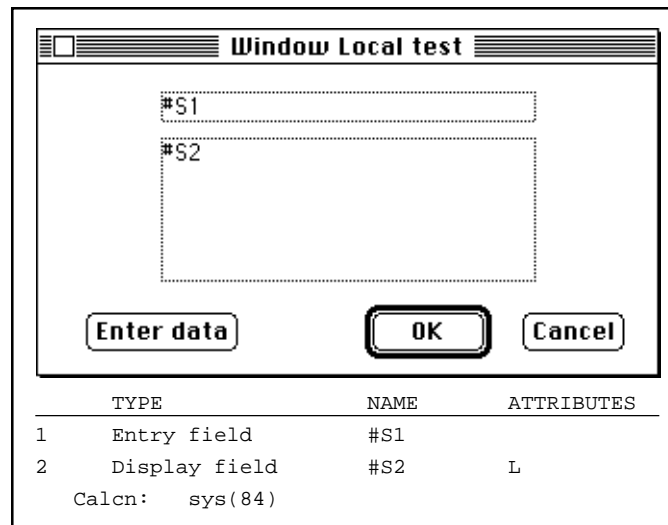


Figure 3 Window for testing event variables with a local display field.

that are executed. We have just examined what happens *before* the procedures execute. The actions of field attributes are “hidden” from the trace mode of the debugger, but they are still available for our use and can be of great benefit to us once we know how they operate.

Now insert the following field procedure for the entry field to redraw the display field when you leave the entry field. You can also turn off the local attribute in the display

would show either #RETURN or #ENTER. The local redraw still takes place if an OK event is precipitated — and keyboard-based events are detectable at this point. But a Cancel event in any form aborts the redraw of local fields.

It is also not appropriate to attempt to detect the #OK or #CANCEL variables (or any other message variables) in procedure statements other than control procedures. Specifically, you can't follow an *Enter data* command with a conditional block like this.

```
If #CANCEL
  commands
End if
```

The #CANCEL will never be true here and the commands that follow it in the conditional block will never be executed. The reason is that the *event variables only have values during the event cycle*. If the #OK or #CANCEL is allowed to come to term and is not redirected by a SNA command, or if a SNA command to perform an OK or a Cancel is carried out, *Enter data* is terminated and the Flag (#F) is either set to "true" ("YES" or 1) or cleared to "false" ("NO" or 0).

Since the message variables are all cleared at this point, testing the Flag is the appropriate thing to do in the procedure block above. It should look like this.

```
If flag false
  commands
End if
```

In some cases, it may be necessary to trap for further conditions in addition to the "flag false" condition. In these cases, we would employ the #F variable. Our compound conditional block would then look like this.

```
If not(#F)& conditions
  commands
End if
```

Cascade of Control

Once all the field attribute effects of the operator's action have been carried out, the appropriate message variables are turned on and the collection of messages is sent "downstream" for processing by the control procedures. These control procedures are executed in an orderly progression as follows: (Please refer back to Figure 2 if it helps you visualize this explanation).

The first procedure to be consulted regarding the current collection of messages is the *field procedure* of the field that was the recipient of the operator's action. The messages are not being "sent to" this procedure, as we commonly say. The messages are merely switched on and the procedure is executed. If there is no procedure for the current field, this level of control is passed over.

Field procedures, when used as control procedures, are the appropriate place to detect and react to field-specific event combinations. For example, if you want to check if the data entered into a field falls within a valid range, you would use a field procedure that reacts to the #AFTER message.

```
If FIELD>500|FIELD<100&#AFTER
  OK message {The value must be
    between 100 and 500.}
  SNA remain on current field
End if
```

Field procedures can be much more complex than this simple example. They can include both fixed and conditional procedure blocks. Different types of fields require different types of procedures.

For example, since we might want to perform different tasks upon entering an Entry field than when exiting it, we should use a field procedure of this form.

```
If #BEFORE& other criteria
  commands
Else if #AFTER& other criteria
  commands
End if
```

In the current version of OMNIS 7, radio button and check box fields can be simpler because only one type of event, a click, can trigger their procedures. Furthermore, the click on these field types can only be detected in *Enter data* mode. It is often not necessary to assign conditions to the procedures for these field types. This may not remain the case in future versions, so you still may want to trap for specific events.

Other conditions you still might consider detecting are the existence of the #SHIFT, #COMMAND or #CTRL, or #OPTION or #ALT modifier messages. These allow us even more flexibility in our applications by permitting different actions to be carried out when the modifier keys are held down. For example, if a shift-click is performed on one of a group of checkbox fields, we could react to that action by selecting *all* the checkboxes.

Pushbuttons work in a similar way, but they really don't enter into our discussion on control procedures. That is because the procedure behind a pushbutton is only executed if that field is *User defined*. The procedures behind any other type of pushbutton (like OK or Cancel) are never executed — unless they are called as subroutines from some other procedure. It is unnecessary to enclose procedures in pushbutton fields with an *If #CLICK* condi-

tional statement. They will, however, react to the modifier messages as with radio buttons and check boxes above.

Since List and picture fields react to either single or double clicks, it is usually necessary to distinguish between these two events in field procedures as follows.

```
If #CLICK
  commands
Else if #DCLICK
  commands
End if
```

Of course, if you only want the field to react to a single or a double click, only the one condition would be required in your field procedure. If, however, you apply no conditions to such a procedure, it will execute twice if the operator double clicks on the field — once for the initial (single) click and once for the completed double click. All double clicks must begin with a single click. This will come up later in the discussion on the Event Cycle.

Even if the field acted upon has no procedure, control is now passed to the current *window control procedure*. This level of control is appropriate for detecting and reacting to event message combinations that span multiple fields, but still refer to a process for the current window. Detecting #OK, #CANCEL, and #CLOSE are common examples of window control procedure tasks, but other message variables (even those specific to field processes) can still be dealt with on this level.

For example, suppose that we need to know, after the *Enter data* mode has been completed, which entry field had last been entered. Since the field number variable, #EF, only has a value during the event

cycle, we need to assign a variable (say, #2) to hold that value so we can use it outside the event cycle. We have two choices for how to do this. We could place the following conditional statement in each entry field on the window.

```
If #BEFORE
  Calculate #2 as #EF
End if
```

This would take a lot of code, since the procedure must be copied into *each* entry field. An alternative is to put the same statement in our window control procedure. Since we want to perform this process for *any* entry field, this is the more appropriate place for this procedure segment — and we only need one copy of the code.

When the *Set window control procedure* command is issued, whatever procedure is specified, regardless of where it resides, becomes the control procedure for the window currently “on top”. Think of this as being the “current” window. The procedure specified will then monitor events for that window. If another window is brought to the top, whether by issuing the *Open window* command or by the operator clicking on it, *its* window control procedure will come into force, replacing that of the window that *was* on top. Think of window control procedures as being “local” with regard to the window, but “global” to the fields on that window.

Finally, the *application control procedure* level is consulted about the current collection of messages. An application control procedure is only necessary if you need to monitor processes that span multiple windows or need to react to a non-window-specific event (like selecting a menu item when in the *Enter data* mode). The most com-

mon situation is where there are a number of open windows, where clicking on a window to bring it to the front has been allowed, and a process that is global to the collection of windows must be performed when a new window is brought to the top. Window-specific needs should be dealt with in each window’s control procedure by trapping the #TOTOP message.

The Event Cycle

The scenario outlined above describes two turns through the Event Cycle. At the end of an event cycle, the “next action” is taken. This will either be the default action required by the event messages that were just processed or the action dictated by the last *SNA* command encountered during the cycle.

Most events triggered by an action of the operator cause more than one event cycle to happen before control is returned to the operator. Consider what must take place in tabbing from one field to the next. First, the cursor must leave the current field. This triggers the #AFTER and #TAB messages *for the field being left* which are processed by the three levels of control procedures in their turn.

Assuming that the default next action is allowed to proceed, the cursor then enters the next entry field. This triggers the #BEFORE message *for the field being entered*. All three levels of control procedure are again consulted about this combination of messages as before.

Only after all necessary event cycles have been completed does control return to the operator. This knowledge is useful to us in structuring applications and determining how to use control procedures to greatest advantage.

Quit to Enter Data

There are times when it is unnecessary to invoke the broader levels of control. Often a set of messages can be dealt with completely by the field procedure and the window and application control levels would just be wasting execution time. We can bypass the succeeding levels of control by invoking the *Quit to enter data* command. This command does not actually cause *Enter data* to occur (as the name seems to imply), rather it “quits” from the normal succession of event monitoring procedures and returns to the main event cycle.

For example, suppose that we have a single character field for inputting gender on a data entry window and we want to trap for inappropriate entry. The only appropriate values for this field are “M” for “Male”, “F” for “Female”, or an empty value indicating an unknown gender. If we detect and react to an incorrect entry, there is no need to consult the broader levels of control. Or procedure would look like this.

```
If
#AFTER&not(#CANCEL)&len(GENDER)
    &not(pos(GENDER,'MF'))
    OK message {Please enter "M"
                for "Male", "F" for
                "Female", or leave blank.}
    SNA remain on current field
    Quit to enter data
End if
```

If the *Quit to enter data* command is executed in a field or window control procedure in response to the #AFTER message, the #BEFORE cycle will still occur as usual. *Quit to enter data* only bypasses the broader levels of control for the current event cycle. If a succeeding cycle is required, *Quit to enter data* jumps immediately to that cycle.

For this reason, the command is normally only appropriate in field and window control procedures. Invoking it in an application control procedure would be a futile gesture unless it were simply desired to skip over the rest of the application control procedure. In this case, *Quit procedure* would serve the same purpose.

If the *Quit procedure* command is issued in a field or window control procedure, the rest of that procedure is bypassed, but the next level of control procedure comes into play as usual.

SNA Commands

SNA (Set Next Action) commands allow us to redirect the normal flow of events. The current set of messages is not effected by *SNA* commands, and the ultimate action is not carried out until the end of the event cycle. A simple experiment to demonstrate this is detailed further in this article.

Only the last *SNA* command issued in an event cycle will be carried out. If a *SNA* command is executed in a field procedure, it can be overridden by a *SNA* in the window or application control levels — or even by another *SNA* executed later in the same field procedure (during the same pass).

A *SNA* command cannot set up an event detectable in the same cycle. Specifically, you can't invoke a *SNA perform an OK* and then detect #OK. The *SNA* doesn't turn on the #OK message, it just redirects processing so that the effect of an OK will conclude the current cycle.

We can demonstrate this with a simple experiment. Create a window like the one in Figure 3 that has two entry fields (I used #S1

and #S2) and the same three pushbuttons (Enter data, OK, and Cancel). Create the following procedures for the specified fields.

```
1 #S1
If #AFTER
    Send to trace log
        {After #S1 sys(84)=[sys(84)]}
    SNA perform an OK
End If

3 Enter data
Trace on
Begin reversible block
    Set window control procedure 240
        {Window control procedure}
End reversible block
Enter data
Trace off
Call procedure *Options/12909
    {Open Trace Log}

240 Window control procedure
Send to trace log
    {Window control
        sys(84)=[sys(84)]}
If #OK
    Send to trace log {Detected #OK}
End If
SNA perform default action
```

Once you have set up the window and saved it, open it, click the Enter data button, tab once, then click the OK button. The trace log will automatically appear to show you what transpired. The entire log is too long to include here, but let me summarize what you will see.

The first event to occur is the cursor entering field number 1 for the first time. Both the field and window control procedure cycled as we would expect.

When we tabbed to the second field, our field procedure sent the original *sys(84)* string to the trace log and then executed the *SNA perform an OK* command. If this truly turned on the #OK message, then

our window control procedure should be able to detect it.

It doesn't. The *sys(84)* string sent to the trace log is identical to the one sent by the field procedure. Furthermore, the *If#OK* condition was not met, so processing skipped to the *End if* statement. Later, when we actually clicked the OK button, this condition *was* met and the text indicating that was sent to the trace log.

If we were now to remove the *SNA perform default action* command from our window control procedure and run the experiment again, we would see that the OK is *executed* after the tab from field number 1 (after the window control procedure runs), but *#OK* is still not detectable within the window control procedure.

Mode Variables

On page 7-25 of the *Application Designers' Handbook*, the use of "state" variables is suggested to help a control procedure react differently when different processes are running. Here they suggest setting up a single character field (assumed to be in a memory only File) to keep track of the current enter data mode ("I" for "Insert" and "E" for "Edit"). I have also seen other people use a longer character field for storing the entire string ("Insert" or "Edit"). While this is a very good and useful idea for helping field procedures sort out the appropriate reactions to events in different modes, let me suggest an alternative method.

I should point out here that we need such variables to determine what "flavor" of *Enter data* is currently operating. We have the *#EDATA* message variable for determining that we are in an *Enter*

data mode, but OMNIS 7 can't automatically set a data entry type mode for us because it can't always know what we intend to do when we invoke *Enter data*.

I use a different variable for each anticipated mode. In my memory only File (you will rarely need more than one of these per application) I have Boolean fields named EDIT, INSERT, and FIND. (I sometimes use *Enter data* mode to emulate a *Prompted find* for the extra control I can have. Control procedures are not in effect during a normal *Prompted find*.)

I treat these variables like the message variables in my control procedures for greater readability and speed of execution. Since they are Booleans, they have no value (not even "NO") until I set one — and I only set the appropriate variable to "YES" (by calculating it to 1) in a reversible block at the beginning of the procedure to which it refers. I can then have statements like

```
If INSERT&#OK
```

which only takes five tokens for OMNIS to represent instead of

```
If (STATE='I')&#OK
```

which takes eleven (and more if you use the full string "INSERT"). See the article in this issue on "eval() Function" and the "Streamlining Algorithms" article in Volume I Number 2 for more on minimizing execution time for expressions. I find that both readability and execution are enhanced by using the Boolean field type for mode variables in this manner.

Using Control Procs

The *Application Designers' Handbook* makes some suggestions on

pages 7-14 and 7-16 for the use of window control procedures that differ from the advice I generally give. I offer you an alternate rationale here.

I only invoke a window control procedure when I need one. I maintain separate window control procedures for different processes (Insert vs. Edit vs. Find). The purpose of this is to keep them as small as possible for faster execution and to enhance the legibility of my code.

If the choice is made to use a single window control procedure to monitor all events that should be trapped at that level, a window control procedure could become quite large and cumbersome in a real world application. Although only one block of commands in a composite conditional block (If...Else if...End if) will be executed on each pass, OMNIS 7 still has to scan and evaluate each condition (if we can believe the trace log). OMNIS executes these quickly, but I prefer to make OMNIS do the minimum amount of work.

The same applies to the work I must do in reading the code. If I am tracking down a change that needs to be made in the Edit mode, but have to read all the conditions from the Insert mode as well, it seems less efficient to me. If I have separate procedures for these two modes, and always put the Insert one as procedure 240 and the Edit one as 239 (or some other convention I will stick to), I can go straight to the one where I need to work.

Neither method is more right or wrong. They are a matter of style and personal preference. Since the manual appeared to be setting down hard-and-fast rules, I thought that presenting this alternative view was important.

New Features of Lists

Lists in OMNIS have been a source of either inspiration and frustration for many OMNIS programmers since the introduction of Omnis 5 in the summer of 1989 on the Macintosh platform—and way back to the introduction of Omnis Quartz on the original Windows platform. They offer us great power and flexibility in manipulating combinations of field values, but it took a while for most old-time OMNIS programmers to feel comfortable with the technology.

OMNIS 7 has now arrived with a new set of more complex (and powerful!) list capabilities. This article is intended to get you on your way to using these new features effectively with a minimum of difficulty.

Much of this material is also given in the manual set that you received with OMNIS 7, but the information is spread across many volumes. This is the nature of manuals. The information is gathered here for your convenience and expanded upon.

List vs. List Field

In this and all discussions on Lists in OmniScience, it is important to distinguish between Lists and List-type window fields. Many people seem to merge the two concepts in their minds—and this can lead to confusion and frustration in day-to-day OMNIS programming. For those who haven't had the opportunity to read "Lists and List Fields" from OmniScience Volume I Number 1, here is a brief summary.

A List is an area in memory used for manipulating a collection of combinations of field values. It is

defined (using the procedure command *Define list*) by the field names assigned to it. These can be thought of as column definitions within the List for easier visualization. The user has no direct view of the contents of a List. (The programmer can, however, both see and manipulate the contents of a List on a cell-by-cell basis using a Field Value window in the Debugger. See the article titled "Adventures with the Debugger in this issue for a further explanation of this feature.)

A line in a List can be thought of as being similar to a record in a File. The field values on a specific line "belong together" as a distinct combination of values. This combination is often an archival or historic view of the CRB, since the most common means of filling a List line with values is by copying the contents of the CRB into that line. You will also occasionally find me referring to a line of a List by the term "record image".

A List-type window field is used to display the contents of a List for the application user and to allow that user to select a line or lines from that List. As with any other window field, *a List-type window field is not the List itself*—it is only a representation of the contents (or part of the contents) of the List. The name of the List whose values are to be displayed is simply one of the attributes of the List-type window field.

A List is *not* defined by the calculation supplied in a List-type window field. This calculation only determine which fields from an already-defined List will be displayed. I emphasize this because many of students have asked for were clarification of the concept.

Each line shown in a List-type window field is a single string formatted to display a combination of field values (a record image). This formatting is most often done using the *jst()* function and a mono-spaced font is used for the List-type window field to columnize the displayed values when more than one field (column) is to be shown.

The only operations that can be performed on a List through a List-type window field are selection operations. All other operations on Lists, such as setting the current List, defining the current List, adding or deleting lines, or manipulating the selected lines must be performed through procedures.

Multiple Selected Lines

The *Application Designers' Handbook* gives an excellent and succinct explanation of how to select lines using the mouse beginning on page 11-15. For reasons of completeness with regard to this article, I paraphrase some of that explanation here.

Only one line in a List can be the current line—that is, #L can only have one value. This remains true for OMNIS 7 as it was for Omnis 5. In Omnis 5, this was the only line that could be considered to be "selected"—and the current line was always selected. Neither of these statements is true in OMNIS 7.

In OMNIS 7, the original selection of a line as the current line—either by calculating a value for #L or by clicking on a line displayed in a List-type window field—both sets the current line number and selects that line. But other operations on selected lines may cause the current line (#L) to not be selected. The variable #LSEL is a Boolean variable that indicates

whether the current line of the current List is selected. Its value can be modified by any number of selection operations, in addition to a direct recalculation.

If a List-type window field is given the *Show selected lines* attribute, more than one line of the associated List can be selected by specific mouse and keystroke combinations using that field. This attribute more fully means "...and allow the selection of multiple lines", which may help you to remember its purpose. In a field so designated, we can select multiple lines from the associated List in many ways.

Clicking on a line and then dragging the mouse up *or* down will select all the lines included in the click-and-drag operation. The line that was originally clicked on remains the current line. The current line is always distinguished as the one with the "marquis" (dotted line) around it. This feature only appears, however, when the List-type field is the current field on the window, so it also indicates that the List-type field is the current field. There is no other visual indication of the value of #L.

Another way of selecting a continuous range of lines is to first click on a line (making it the current line) and then to shift-click on another line. All lines from the current line to the shift-clicked line will be selected, whether the shift-clicked line is above or below the current line. The line receiving the first click remains the current line.

A discontinuous group of lines can be selected by command-clicking on additional lines (control-clicking in Windows — the control key on the Macintosh does nothing here). You can also command-click-and-drag to add a continuous group of lines to the current selection. The line receiving the command-click becomes the current line.

Command-clicking a line that is already selected deselects that line, but also makes it the current line (sets #L to that line number). This

Lines that do not match the current search specification can be stripped from the selected set with the *Deselect non-matches (AND)* option. Since this command only effects currently selected lines, it can be sped along by also using the *Only test selected lines* option. As with the *Select* option, scanning of List lines begins with the line after the #L except as noted above.

To ensure that *only* lines that match the current search specification are selected, either use *both* the *Select*

and *Deselect* options or precede the *Search list* command with a *Deselect list line(s)* command with the *All lines* option.

None of the multi-line processes mentioned above effects the value of #L. The current line remains current, but its selection state may change. The *Search list* command *without* any options selected, however, *will*

select a single line *and* make it current.

Selected vs. Saved

Once a set of lines in a List has been selected, we have the option of "saving" the selection status of these lines, making a different selection, and performing various operations on the two sets. We speak of the "current" selection as the *selected* set and the previously selected group as the *saved* set. These are also the terms used in the procedure commands that operate on these sets. If no selection has been saved, the saved set is empty. If no selections have been made, or if all

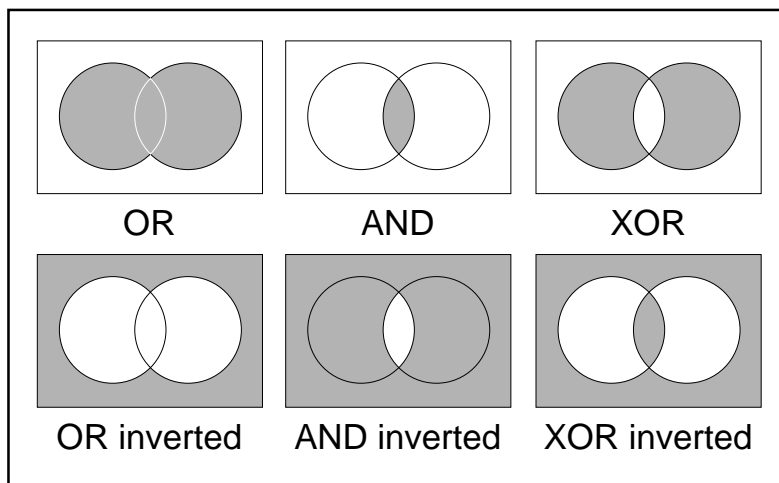


Figure 4 Diagrams of Boolean operations involving two subsets of a global set.

is but one of a number of ways in which *the current line may not be "selected"*.

Search and Select

We can also select or deselect multiple lines by using the *Search list* procedure command with some of its options. The *Select matches (OR)* option adds all lines that pass the current search specification to the group of selected lines if either it is accompanied by the *From start* option or if there is no current line (#L=0). If *From start* is not selected, the process of selecting matches begins with the line *after* the current line.

lines have been deselected, the selected set is empty.

Lines are added to the saved set using the *Save selection for line(s)* command. By default, this applies only to the current line, but a line number can be specified (explicitly or through a calculation) or the *All lines* option can be employed. Once lines have been saved, an alternate selection can be made and operations can be performed on the combination of those two states.

“Saving” does not imply writing anything to disk. Selecting and saving of List lines all occurs in RAM. If, however, a List is stored in a record, both the saved and selected states for each of its lines at the time the *Update files* command is executed are also stored.

The *Restore selection for line(s)* command copies the saved state of the specified line or lines into the selected state. Both states are then the same. This command also has an *All lines* option.

There is no simple test we can employ in the current version of OMNIS 7 to determine whether a line is saved. That is, we don't have a variable comparable to #LSEL to indicate the saved state of the current line. If we need to know whether the current line is in the saved set, we can detect this in the following way. Create a Boolean field in a Memory-only File with the name LSAVED. The following procedure will give it a value for the current line (#L).

```
Swap selected and saved
Calculate LSAVED as #LSEL
Swap selected and saved
```

This test is only valid at the time it is made. It does *not* have the same power as a system variable would.

The *Swap* command also allows us to specify a line and has an *All lines* option. This command merely causes the “selected” and “saved” states for the specified line(s) to trade places.

The *Save* command effects the saved set only, the *Restore* command effects the selected set only, and the *Swap* command effects both. All three — and all the commands we are about to explore in the next section — effect only the current List.

Boolean Operations

More complex operations can also be applied to create various combinations of selected and saved states. Each of the commands in this section effects *only* the selected state for the specified line(s). That is, a *new* selected set is generated based on the old selected and saved sets and the operation performed.

You may find the diagrams in Figure 4 helpful in the discussion that follows. In these diagrams, the rectangle represents the entire set of records contained in the current List. The lefthand circle represents the saved set and the circle on the right represents the selected set. The shaded area indicates the *new* selected set generated by the command labeling the rectangle.

The diagrams show the result of using the *All lines* option with these commands, but for a specific line, you simply need to determine to which region the line belongs to see the resulting selection state that operation will give it.

The one operation not shown diagrammatically is the *Invert selection for line(s)* command. This simply toggles the selected status of the specified line(s) between se-

lected and not selected. A diagram of this would show the right circle (the previous selected set) as white and the rest of the diagram shaded.

The *OR* and *AND* operations are the easiest to explain. A line will be selected by the *OR selected and saved* command if it was *either* selected or saved (or both) before the operation. A line will be selected by the *AND selected and saved* command if it was *both* selected *and* saved before the operation. Note the effect of applying the *Invert selection for line(s)* command after these two.

The *XOR* or exclusive OR concept is a little tougher to get across. A line will be selected by the *XOR selected and saved* command if it was *either* selected *or* saved *but not both* before the operation. If a line is *both* selected and saved or if it is *neither* selected nor saved, it will not be selected by the *XOR* command. Such a line *will* be selected by following the *XOR* with an *Invert* command.

List Command Options

Many Omnis 5 List commands have new options and more power in OMNIS 7. This is another example of the evolving nature of OMNIS.

Add line to list and *Insert line in list* are now virtually identical except for their default states. *Add* appends a line to the end of the current List by default while *Insert* inserts a line at the current line by default. Both commands allow a line number to be specified.

The more impressive aspect of these commands is the new facility to optionally specify a comma-delimited list of values to be carried to the specified line. The values list has one restriction — the values

are mapped to the fields of the List in the order they appear in the *Define list* statement. Any value left blank will generate the appropriate type of null value for the target field type.

The *Replace* command has a similar facility. It allows us to replace a single “cell” or selected “cells” in the specified line of the current List by using the value list facility. In this case, though, if we leave a value slot empty, the current value of the corresponding field in the specified line remains untouched rather than being given a null or empty value.

To replace a field with a null value, we simply have to supply the appropriate value (“ for strings, 0 for numbers, etc.). To replace a Boolean field with a null, use a field name from a Boolean field that is known to be null, like one of the mode flags from a Memory-only File. Attempting to calculate a Boolean as null by using a null string (“) will generate a “NO” value because the Boolean equivalent of a null string is “NO”.

Redefining Lists

The *Define list* command also clears the current List of all lines and values. In general, this makes logical sense, since there is no guarantee that the old values will fit the new definition. However, there are times, especially when using OMNIS 7 as a SQL front end, when we might want to remap the values in a List to new field names.

This is the purpose of the *Redefine list* command. It is the responsibility of the programmer to use appropriate field names in the redefinition. OMNIS 7 simply changes the names of the columns, *not the data type*. For example, if you re-

place a numeric field name with the name of a string field and then add or insert more lines, OMNIS will do its best to convert the string coming from the CRB to numbers — but any string that doesn’t look like a number will be converted to a zero.

The fields of a List are redefined in the same order in which they were originally defined. If you leave some of the slots in the field name list blank, OMNIS will just use the existing name. This allows us to redefine selected column names.

Popup Lists

We now have some alternate ways of displaying and using Lists on a window. Our only choice in Omnis 5 was a List-type window field, but we have two new options.

The first of these is a Popup List field type. This type of field comes from the Macintosh platform, but is available to both Macintosh and Windows users of OMNIS 7. The normal use of such a field is to make a mutually exclusive selection of a line from the corresponding List. This field type usually only displays a single column of values. Traditionally, this type of field uses the Chicago font on the Macintosh and should use the System font in Windows.

If the *Show selected lines* attribute is turned on, and if multiple lines have been selected in the corresponding List, the selected lines are displayed with check marks to the left of the field values.

This field type has the advantage of displaying a blank value when no value has been selected from the corresponding List. Since the field is “collapsed” when not being manipulated by the mouse, this

blank state is very noticeable to the operator. On the other hand, this field type has the disadvantage of *requiring* the use of the mouse to select a line. Its use may not be appropriate or desirable for some data entry environments.

Combo Boxes

From the Windows environment comes an alternative to the Popup List field called a Combo Box. The reason for the name will become apparent in the next paragraph. This field type has a similar purpose to that of a Popup List field, but the implementation is quite different. It also collapses to a single line when not being manipulated by the operator.

A combo box can be “opened” by clicking on the arrow button next to the field display or by tabbing into the field and typing the space bar. Tabbing back out of the field collapses the List again.

This field type has the advantage of acting like both a List-type window field and an entry field. It can be entered by tabbing or clicking, generating a #BEFORE message. It can be left by tabbing or clicking, generating a #AFTER message. It can be navigated in the same ways as a List-type window field by using either the arrow keys or keystrokes with wildcards as well as being scrollable by the mouse.

Combo boxes have the disadvantage of displaying the contents of the first line in the List whether a line has been selected or not. It is also not possible to deselect a line (make #L equal to 0) through keystrokes — the mouse must be used to click on the blank line at the bottom of the List. Again, your use of this field type depends on your data entry needs and preferences.

List-based Reports

We can now base reports entirely on the contents of a List. This message is primarily good news with one minor caveat. The good news is the very ability to perform this task and the speed and flexibility with which OMNIS 7 handles the job. Simple, fast reports directly from Lists operating entirely from RAM can have a number of practical uses.

The old Omnis 5 restriction of a maximum of 30,000 lines in a List has disappeared. OMNIS 7 is only limited by RAM as to the total number of record images a List can contain. List contents can be sorted and subtotaled on up to nine levels, just like CRB-based reports. If the reporting is done on a local printer, absolutely no network traffic is generated by the List-based reporting process!

The only limitation to List-based reporting is that *all the fields for the report must be contained in the List*. This is because the values are coming from the List buffer and the CRB remains static during the reporting process. It may be only in the original release of the product or it may be a permanent feature, but *Automatic find* fields do not retrieve records in List-based reports. Fields outside the definition of the List used for the report *will* show their current (static) CRB values, but there is no way to manipulate them or relate them to List values.

There is nothing wrong with OMNIS performing in this way — it actually makes perfect sense for it to do so. But the uninitiated programmer may be confused at first about the *Automatic find* situation. Once you are aware of the rules, the job is straightforward.

To set up a List-based report, a Report Format must be created. In the Sort Fields window of the Report Format, click on the List-based report check box and supply the name of the List on which the report is to be based. Make sure that all the fields used in the report are included in the definition of that List when the report is to be run. Then just use the *Print report* command as usual.

For information on using a List as a sort buffer for complex, CRB-based reports, see the article titled “The Omnis Report Generator” in OmniScience Volume I Number 1, pp. 26-29.

Storing Lists

In OMNIS 7, we now have the ability to *store* Lists in our datafile. There is a new List field type available for defining fields in File Formats. There are a variety of uses we can make of this feature, but we need to always keep a watchful, practical eye on our imagination — some of our ideas may have drawbacks or cause us some serious problems down the road.

One possible use for stored Lists is to store “lookup tables” that have multiple values. I’m not talking about lists of customers or part numbers, those may change frequently and there are better ways of handling them when you have a large number of records to deal with. I am speaking more of lists of shipping charge rates, tax tables, and other static background data that rarely require maintenance but that are often consulted in day-to-day operations.

In older versions of OMNIS, we had to set up an entire array of fields in a Constants File and work out a lookup scheme using the *fld()*

function. While this is still a valid method — and may be preferred in some cases — stored Lists seem to have great potential in this area.

Another possible purpose for List fields for storing actual data is the use of multi-valued fields in records. A simple example of using a List field in a record is a field in a Customer File for storing multiple telephone numbers. In this era of modern telecommunications, many busy people seem to have an unending list of possible telephone numbers — one or more office, home, fax, mobile phone, or pager numbers can apply to any of our clients. A List field defined to include country code, area or city code, telephone number, and type of telephone could be just the thing for keeping track of this growing list of telephone information.

Each List stored in a record contains the List definition, the selection states of each line, and the values of the List variables (like #L) in addition to the field values contained in the List. This means that, when a new record is created, the List must be defined at the same time that you would normally calculate default values for fields.

There are three sources that you can use for List definition fields for such a List — fields within the same File Format as the List field, fields from a Memory-only File, or Format Variables. Using the latter will require that you use the *Redefine list* command whenever you need to access the contents of this field outside the Format where the records were originally entered, so you may or may not find this to be a practical method. Using fields from a Memory-only File will allow global access to these field names throughout the application, but exported records will have the same

problems as above if import into an application where those field names don't exist. Using field names from within the same File Format will make those fields available on import as long as those field names are still used, but it seems like a lot of extra baggage to carry around.

Stored Lists in Reports

If we have stored Lists in a File Format List field, we may very well want to print the contents of that field on a record by record basis in a report. The problem — there is no List-type field available in Report Formats. Contents from a List field in the CRB don't map directly to *any* of the field types available in Report Formats. Since this List is only *part* of a record, a report *based* on the List isn't appropriate either.

As OMNIS continues to evolve, such facilities will appear and will undoubtedly have more flexibility and power than we can imagine today. But for now, we need to find some other means of retrieving those values in a form the Report Format can handle.

The problem can be demonstrated by the telephone number example I mentioned earlier. It is very convenient to store telephone numbers in this way, and they can easily be looked up while sitting at the computer. But an address book report would have a difficult time extracting and printing the values. There are two solutions that I envision, both having their respective strong and weak points.

The first solution uses a number of no-name fields in the Report Format on many different lines. A no-name field on a Report or Window Format is a Character field with a maximum length of 32,000 charac-

ters. In this example, we have many choices as to how to use these fields. We could use one no-name field for each column in the List on each report line, or we could use one no-name field and calculate its value with the *lst()* function, or some configuration in between.

The values generated for these fields would be derived using the *lst()* function. The calculation for fields on different lines of the Record Section would reference the corresponding line in the List. The Telephone Type value, for example, from the first line would be derived by this formula.

```
lst(TELEPHONES,1,TYPE)
```

We would also need an *Invisible* no-name field to hold the concatenation of all columns for the appropriate List line on each Record Section line. This field would also be given the *No line if empty* attribute. In this way, only List lines that contain values would generate report lines.

Another line-suppression method you might be tempted to try would be to calculate the value *#LN >= line number* in a *No line if empty* field on each line as a cutoff criterion. First, you must use a numeric field for this instead of a no-name field because a zero value is not considered to be "empty" in a string field. But there are still two problems. The first is that this method won't skip lines that are blank. The second is that there is no assurance that TELEPHONES is the current List, so #LN may not have the "correct" value.

In any event, the drawback to this method is that it requires that you decide on the maximum number of lines you ever expect TELEPHONES to contain (is ten

enough?). The report format may perform a little sluggishly when forced to evaluate a number of fields it won't be using.

The advantage of this method is that it can be used with either the *Print report* or *Print record* method of printing.

In my other technique, we must print our report using the *Print record* command in a *Repeat* loop. For each record to be printed, we will call a subroutine that will load the values in each line of the List into a long string field (say, #S1). The *lst()* function is used to format each line image and a return character, *chr(13)*, is inserted before each succeeding line after the first. The procedure would look like this.

```
Set current list TELEPHONES
If not(#LN)
  Quit procedure
End if
Calculate #S1 as ''
Calculate #L as 1
Repeat
  Calculate #S1 as
    con(#S1,mid(chr(13),#L<>1,1),
      lst(CNTRY_CODE),-3,
      lst(AREA_CODE),-4,
      lst(PHONE),-10,' ',2,
      lst(TYPE),10))
  Calculate #L as #L+1
Until #L>#LN
```

Notice the use of the second parameter of the *mid()* function as a switch to turn off the carriage return character for the first line.

We would place a field for #S1 on the report and make it an Extending field two lines long. It would then require as many lines for each record as the corresponding List.

In the next issue, I will discuss more storage, retrieval, and reporting techniques using Lists.

Adventures with the Debugger

As our applications become more complex, it becomes increasingly difficult to keep track of what is happening in OMNIS (or what *should* be happening in OMNIS) in our heads. We need to understand in what order our procedure commands are being carried out and what changes are occurring to the values of certain fields and other states of the application as processing proceeds. It is also helpful to be able to halt processing and look around at these values — even make changes to them if necessary — and then resume execution from the point the procedure paused. These are some of the many things we can do with the new OMNIS 7 Debugger features.

If you have ever written programs in another language that has a debugger, some of what OMNIS 7 offers will be familiar. I have seen my favorite features from many other languages built into this debugger, along with many new ones that are specific to OMNIS.

This aspect of OMNIS 7 is truly a remarkable piece of programming!

The Debugger is probably the most thoroughly documented feature of OMNIS 7. Perhaps this is due to the excitement of finally having such a feature in OMNIS, or to the excitement generated by its wealth of options and tools — but there are still some things you may have missed in reading the manual. Here is another look at the Debugger.

Option-Click Gateway

One of the many impressive features of the Debugger is the ability

it gives us to navigate through the application in “Design” mode. We can option-click on the name of any field or any Format *anywhere* in design mode (even in literal strings!) and a menu of options magically pops up. These options allow us to either peek further into what OMNIS is doing with our application or to hop to another part of the application in Design mode directly to examine or modify something there. The option-click feature reminds me of teleportation gateways depicted in many science fiction movies.

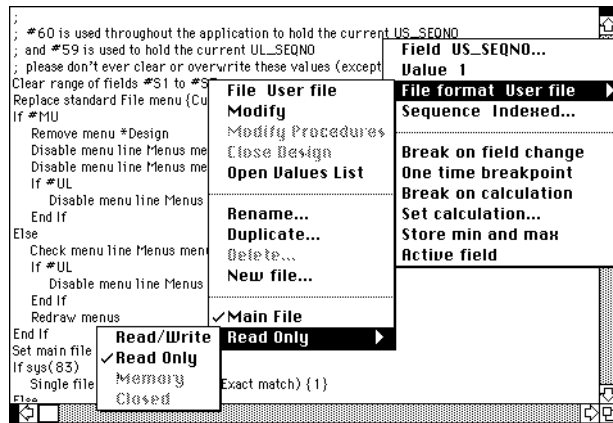


Figure 5 Menu available option-clicking on US_SEQNO field

In Figure 5, I show the result of option-clicking on the name of a field that is referred to in a comment and then following all the submenu paths available. Pretty impressive! The first submenu is also the basic menu OMNIS would present if I option-click on the “User file” File Format name. So access to a field in this way is also access to the File Format to which that field belongs.

Look at the options the basic field menu provides! It first confirms the name of the field and displays a summary of the current value. It also shows the name of the File to

which the field belongs and the field’s data type. Selecting the data type line from this menu takes us *immediately* to that field’s definition in the corresponding File Format modification window, allowing us to change the data type or whatever other task we might need to perform there.

Here is one way I take advantage of this feature. For Integer fields that are to store radio button values, I put the translation table (used by the *Lookup table* attribute of a window or report display field) for the expected values in the field description in the File Format. Whenever I need that string, I just option-click on the name of the field, select the data type option, copy the description of the field, and paste it into the Lookup table. This type of technique lends great efficiency to the mechanics of programming OMNIS 7 — thanks to the Debugger!

Selecting the field menu line that displays the current value does nothing and takes us no further. But selecting the first line, the one that confirms the name of the field, opens the Field Value window for that field. A feature of this line that indicates there is another window to be seen is the ellipsis (“...”) character following the title of the option. Any menu item that brings up a dialog box with more information or choices should end in an ellipsis — including in the applications you write.

In the File Format submenu, there is also a line titled *Open Values List*. This opens a window that displays the current values for *all* fields in that File. We will explore some uses of the Values window in the next section.

Option-clicking on Format names is also rewarding. A menu similar to the first submenu shown in Figure 5 appears. The options are nearly identical to those in the top two sections, no matter what type of Format is selected. The first line identifies and shows the type of the selected Format. The next line, *Modify*, opens the selected Format in modification mode on top of any open windows. The *Modify procedures* line only applies to Window Formats in the current version of OMNIS, and it takes us directly to the procedures layer of that window, bypassing the layout view. *Close design* is only available if the selected Format is currently open in modification mode.

The next line is where the greatest type-to-type variation occurs. This line allows us to “operate” the Format. For Window Formats, this lets us *Open* that window in user mode. For Menu Formats, it lets us *Install* the menu. It also lets us *Print* a Report Format and set a Search Format as the *Current search*. For File Formats, it will *Open Values List* as shown.

For *all* Format types, the second section of this menu is the same. We can *Rename...* the selected Format, *Duplicate...* it, *Delete...* it, or create a *New* Format of the same type. These are powerful features indeed when you consider that we used to have to go all the way to the Design window, select the proper type of Format, select the Format we want to operate on, and then perform the operation. Now we can do this “on the fly”! And to duplicate or delete a File Format, we used to have to go to the Application Utilities area after mounting the Utilities menu — and *that* action puts away all our windows and removes any installed menus. This new functionality is *very* nice!

There are other places where we can option-click to gain more access to things and I will mention a few more later in this article. But be adventurous — try option-clicking on things in the design mode that look interesting and see what might happen!

Field Values

When we discuss the Current Record Buffer in classes, my students often ask if it is possible to “see” the CRB. Until OMNIS 7, the answer has always been “No”. But the OMNIS Debugger now allows us to not only *see* the values in the CRB, but to *manipulate* them as well! This is the purpose of these “Values” windows.

The Value window for an individual field can be expanded and scrolled to accommodate the largest possible field values. The Values window for a File shows each field as a separate line, but a “Full Value” window can be opened for that line by double clicking on it. The Value window for an individual field and the Values window and subsidiary Full Value windows for a File Format can be used to modify the CRB values for the displayed field(s).

If the field in question is a List field, the Value window for that List is truly remarkable. We have cell-by-cell access to each value held in the List. Double clicking on a cell opens a Value window for that cell, labeled as it would be referenced by the *lst()* function. A List field in the Values window for a File Format only bears a notation indicating the number of lines in that List, but double clicking opens up a Full Value window that also offers cell-by-cell access to values.

The current line of the List is marked by a “C” and each selected

line is marked by an “S”. Option-clicking on the line number causes yet *another* menu to appear. This menu allows us to select or deselect that line or make it the current line. We can also clear or delete that line or insert another line *before or after* the one we are investigating. It is truly difficult to contain my excitement for some of these features while writing about them! It seems that this cell-by-cell access to values in a List may have some future implications...

Execute vs. Go

One thing I always wanted in Omnis 5 was to be able to test procedures and subroutines in a Window Format by executing them from the procedure modification window. In the past, I had to copy and paste the procedure to a Menu Format, since we *can* do this with procedures there with the *Execute* command. This command is not available in the Procedures area for Window Formats.

With the Debugger, we have an alternative. We can toggle on the Debugger window overlay for the window procedures using command-Y. We can then specify a *Go point* and issue a *Gocommand* from the Debug menu or by clicking on the *Go* button on the Debugger menu bar. If we select the first line in the procedure as our *Go point*, the *Go* command is the equivalent of the *Execute* command — but we can choose to *Go* from *any* line in *any* procedure.

Selecting a *Go point* is as simple as double clicking on the desired procedure line in the Debug window. The *Go point* is marked by a “G” in the space to the left of the procedure commands in the Debug window. This area is known as the “command margin”. We will be

dealing more with the command margin later in this article.

Once the procedure begins executing, the *Go* button becomes the *Stop* button. We can *Stop* execution at any point by clicking the *Stop* button or by performing the keyboard equivalent. In Windows, the equivalent is pressing the Escape key. On the Macintosh, the equivalent is the Command-Option-Delete key combination (which has a whole different meaning on a Windows machine!).

Stop pauses execution to allow us to examine the current state of field values, etc. If a value did not change as expected, we can change it in a Value window at this point. The procedure line that was next to be executed automatically becomes the current *Go* point. When we have finished exploring and correcting values, we can click the *Go* button and the procedure will continue from where it paused — or we can double-click on some other line and *Go* from there!

Breakpoints

Procedures execute so rapidly in OMNIS 7 that it is nearly impossible to use the *Stop* button accurately when a procedure is running in the *Go* or *Execute* mode. To halt the execution of a procedure at a specific point, we can set a “breakpoint”. When OMNIS encounters a breakpoint, it automatically pauses the procedure as if we had clicked the *Stop* button.

There are different types of breakpoints and they are set up in different ways. The two basic categories are procedure line breakpoints and value breakpoints. The simplest of these to explain are the procedure line breakpoints, so let’s start there.

There are three types of procedure line breakpoints: One-time, Full, and Permanent. A permanent breakpoint is established by inserting a procedure line that contains the *Breakpoint* command. Procedure execution will always halt when this command is encountered. The procedure line after it becomes the current *Go* point. This breakpoint will remain in position until that procedure line is removed or the command on that line is changed.

One-time and full breakpoints are set in the Debug window. To set these breakpoints, the procedure line where the break is to occur must first be selected. Then the appropriate breakpoint type is selected from the “Breakpoint” menu of the Debug window. An alternative method of setting a procedure line breakpoint is by first option-clicking on the command margin next to the desired procedure line and then selecting the breakpoint type from the menu that appears there. The second method is often more convenient. One-time breakpoints are denoted by a “b” in the command margin, while full breakpoints are denoted by a “B”.

In operation, a one-time breakpoint vanishes once it is triggered. The procedure pauses *before* that command line is executed and that line becomes the current *Go* point. Full breakpoints work in the same way, but the line remains a breakpoint after it has been triggered.

Remember, when a line is marked as a breakpoint in this way, the procedure pauses *before* executing the breakpoint line. The *Breakpoint* procedure command *must be executed* to cause a break to occur. Think of that command as applying to the line that follows it if you need a common reference.

Both the One-time and the Full breakpoints are cleared by the *Clear breakpoints* command from the Breakpoint menu or when the application is closed. In the latter sense, they are *both* temporary breakpoints. The *Breakpoint* command is not effected by either of these occurrences.

Value breakpoints pause the procedure whenever a specified value changes. Think of this as a sort of “execute until...” criterion. There are two types of value breakpoints: field breakpoints and calculation breakpoints.

A field breakpoint is assigned to a specific field. It causes any procedure to halt when the value of the field changes. This type of breakpoint is set using the *Break on field change* selection from the popup menu accessed by option-clicking on that field’s name. It can also be set from the “F” menu available in that field’s value window.

Field breakpoints can be set as one-time breakpoints by toggling on the *One time breakpoint* option from either the field option-click menu or “F” menu.

A calculation breakpoint is also assigned to a field, but it doesn’t have to refer to that field within its expression. It is set using the *Break on calculation* option from the option-click or “F” menu. The calculation that will cause the break is set using the *Set calculation...* option.

This type of breakpoint halts processing whenever the calculation *becomes true*. This is an important point. If the calculation is *already true*, the break can’t occur until the calculation first becomes *false and then true*. Becoming false sets the snare — becoming true triggers the trap.

Both field and calculation breakpoints are all cleared by selecting the *Clear field breakpoints* from the Breakpoint menu. Individual field or calculation breakpoints can be cleared by toggling off the *Break on field change* or *Break on calculation* selection. All are cleared when the application is closed.

The current field and calculation breakpoints can be viewed in the Values List windows of their respective File Formats. Field breakpoints are denoted by a “B” next to the field name. One time breakpoints are denoted by a “b”. Fields that are the guardians of calculation breakpoints are marked with a “C”.

When a procedure pauses because of a breakpoint (or, for that matter, because of an error), a message is sent to the “Status line” on the Debugger window overlay. One time and Full breakpoints send a message that simply says “Breakpoint”. The message for the *Breakpoint* command is “Breakpoint command”, but we can append our own message to that which shows up in curly brackets. Our custom message, like any string parameter in OMNIS, can include square bracket notation, so we can also send values to the status line using this feature of the *Breakpoint* command.

A break that was triggered by a change in the value of a field is accompanied by a message that states “Break on field change” and gives the name of the field in parentheses. The name is important because we may have a number of fields set to trigger breaks. If more than one field triggers a break at the same time, only one will be reported in this message. A break triggered by a calculation

breakpoint sends the message “Break on calculation” followed by the name of the field set to tend that calculation in parentheses.

If an error occurs during the execution of a procedure in single user mode, the Debugger window will appear as though a breakpoint had been encountered. The *Go* point will be the procedure line that OMNIS had a problem with, and the status line will contain a message briefly describing the error.

Tracing

Viewing the contents of the CRB at a pause point during procedure processing is only one aspect of debugging. Another equally important aspect is tracing a procedure to see what lines were actually processed. We can learn a lot about the operation of OMNIS — in both standard procedure processing and event management — from this versatile feature.

To trace a procedure, we have a number of options. From the Debugger window, we can either select *Trace* from the Debug menu or we can alter the *Go* button and turn it into a *Trace* button. This button can actually take on four faces: *Go*, *Step*, *Step over*, and *Trace*. If we option-click on the button, it will cycle through these faces in this order. If we *shift*-option-click, the button cycles through its faces in *reverse* order. (Often in the OMNIS design mode, holding down the Shift key while performing an operation means “do the opposite”.) Therefore, if the *Go* face is currently showing, we can go directly to *Trace* by *shift*-option-clicking the *Go* button.

Trace executes a procedure from the *Go* point, but it also shows us exactly which procedure line is be-

ing processed *as the procedure is being executed*. Each line is highlighted as OMNIS processes it. We can watch OMNIS skip over conditional blocks when the condition is not met, cycle through loops until they are exited, jump to subroutines, and even execute control procedures. If you have a large enough monitor on your computer, you can move the Debugger window off to the side and still see it trace as you operate on another window! Which ever procedure is currently being executed shows in that window — even if the procedure is in a different Format!

Tracing a procedure slows execution time somewhat because of all the screen redrawing that must go on, but this is good if you want to follow along. If your machine is fast enough, however, it still may be difficult to follow all the details — and it will certainly be difficult to *remember* them. OMNIS does have a facility for remembering each line that was executed so that you can follow along at your own pace. This facility is called the “Trace Log”.

The Trace Log

As OMNIS traces a procedure, it sends information about each line it executes to an area in RAM called the Trace Log. The basic information sent is the Format name and procedure number of the current procedure and the exact wording of the line being executed.

The Trace Log “remembers” all executed lines up to a maximum limit. By default, the maximum number of lines in the Trace Log is 200, but you can increase or decrease that number as you need. Just remember that the more lines you accumulate, the more memory is required. If the limit is exceeded,

lines will be lost off the top of the Trace Log.

We gain access to the Trace Log by selecting the *Open Trace Log* option from either the Options menu on the Debugger window overlay or the Debugger Options submenu of the Design menu. We can browse through these lines to examine in detail what has transpired during execution and we can use the information provided to navigate through the application to make corrections as necessary.

As in every other area in Design mode, we can option-click on field and Format names shown in the text of the Log. This allows us to correct values on the fly as we test our application. We can also double click on any line shown in the Log and make that the current line in the Debugger window. This allows us to go directly to *any* executed procedure command and change it if it needs correcting.

If we need to archive a copy of the Log, we are provided with a *Print* button that will send the Log to the current report destination. If we want to start with a fresh slate, we also have a *Clear* button.

Breakpoints also apply to *Trace* mode as they do to *Go* mode. The procedure pauses and allows us to look around. The Trace Log is available at a break, and you will notice that the same messages that are sent to the Status line for breakpoints and errors are sent to the Trace Log.

We can also send special information and messages to the Trace Log at any time during the execution of a procedure. There are a number of procedure commands that allow us to do this. The most generic of these is the *Send to trace log* command.

This allows us to programmatically send a string to the Trace Log that can be up to 255 characters long *and* contain square bracket notation. The square bracket expressions can be anything, but we are most likely to specify field names (to show their current value *at this stage of execution*) or *sys()* functions (to show the current state of the application). We can format this nicely with the rest of the string for easy readability.

There are also a number of *Field menu command* options that send information to the Trace Log. One of these options is *Send value to trace log*. We specify one or a number of field names and the current value of each is sent to the Trace Log. A separate line is added for each field, though, so we need to either be careful about overusing this feature or give the Trace Log a greater maximum number of lines.

The option-click field menu gives us an option to *Store minimum and maximum* for that field. The current minimum and maximum value that field has taken on since the option was set are shown at the bottom of that menu, but we can also send either the minimum or the maximum or both values to the Trace Log at any time within an executing procedure by using the respective *Field menu command* options that perform these services.

Another great thing about the Trace Log is that we can send messages or values to it *without being in Trace mode*. Any time we execute any of the above commands, the specified information is sent to the Trace Log. In this way, we can create a “messages and values only” trace of our procedure, which operates faster and requires less memory. If all we need is a view of how values change, this is ideal!

We can also perform partial traces using the *Trace on* and *Trace off* procedure commands. Often we won't need to see a trace of every last command that OMNIS executes. We may only have a small trouble spot that concerns us, but a long procedure that must set things up. In these cases, we would insert the *Trace on* command just before the lines of interest and the *Trace off* immediately after them.

You can do this many times within one execution cycle — but if you do, be sure to send appropriate messages indicating what part of the procedure is currently being traced. If you don't, you can still find your way around, because the *Trace on* and *Trace off* are both sent to the Trace Log.

Active Fields

If we are interested in the values of a number of fields while debugging some part of our application, it can become cumbersome to be constantly option-clicking on lots of field names to open their Value windows. The people at Blyth Software recognized this and gave us some other options. One of these is the ability to declare a number of fields are “Active Fields”.

They also gave us a number of ways to make this declaration. The most straightforward is to toggle on the *Active Field* option in the field option-click menu or the “F” menu in the field's Value window. We can also issue the *Field menu command* option to *Add to active fields list* one or a number of fields. In addition, any field that has been set to *Store minimum and maximum* or any field with a field or calculation breakpoint currently set is already considered to be an Active Field, since we have shown that special interest in them.

So what does this buy us? Well, at any break, we can select the *Open Active Fields List* option from the Options menu in the Debugger window overlay or from the Debugger Options submenu of the Design menu. This will open a window that displays the current values of each of these fields. It's like the Values window for a File Format, but it shows the collection of fields of current interest to us rather than simply all those from a specific File. What convenience!

Fields can also be removed from this list by issuing the command that "undoes" the command that added that field. These, again, are all options of the *Field menu command*. We can issue *Do not store min & max*, *Clear break on field change*, *Clear break on calculation*, and *Remove from active fields list*.

Value windows for individual fields and Values windows for specific File Formats can also be opened programmatically with options from the *Field menu command*. These options are *Open value window* for individual fields and *Open values list* for Files. Both of these options accepts a list of field names. For the first option, a Value window is opened for each field listed. for the other option, a Values window is opened for the Files to which each field belongs. It is only necessary to specify one field per File — only one window will be opened per File in any event.

When issuing commands that can open multiple windows, bear in mind that there is a limit to the number of open windows. The current limit is 24 in Windows 3.0 and 36 on the Macintosh.

The Active Fields List can also be opened programmatically, but to do so , you must call the *Open*

Active Fields List procedure from the Options menu. The Trace Log can be opened in a similar way.

Stepping

The *Step* option from the Debug menu works like a *Trace* with a breakpoint set on each line. The *Step* face can also be given to the *Go* button by option-clicking. It executes only the next line (the one set as the *Go* point) and sends that line to the Trace Log along with any messages that OMNIS may interject. If another procedure is called as a subroutine — even if it is in a different Format — it is shown as the current procedure in the Debugger window.

Another form of the *Step* command is *Step over*. This option works the same as *Step* with one important difference: If *Step over* encounters a *Call procedure* command, it doesn't jump to that procedure. Instead, it allows the entire subroutine (and any of its subroutines) to execute until control is passed back to the current procedure. Executing the *Step over* command on a *Call procedure* line is the same thing as saying "perform the subroutine specified".

Procedure Stack

When a breakpoint or an error is encountered during the execution of a procedure called as a subroutine, we can trace the return path by examining the "Procedure Stack". This is a list of all the called procedures that lead back to the originally executed procedure. The Stack menu on the Debugger window overlay allows us to examine and manipulate the Procedure Stack. The bottom section of this menu shows the currently pending procedures — the one where the break occurred is the one on the

top, the one that called it is just beneath that, etc.

Selecting one of the calling procedures in the stack will display it in the Debugger window. If you hold down the Shift key while making this selection, a new Debugger window will appear on top of the current one if the selected procedure is in a different Format than the current one. You will notice the letter "R" marking the command immediately following the *Call procedure* command. This marks the "Return" point. When the called procedure is completed, processing returns to this command and execution resumes.

The *Clear procedure stack* option is used to ignore the lower levels of the procedure stack and clear all the return pointers. Use this with caution, since any states set in reversible blocks in any of the calling procedures will also be forgotten.

Debugger Options

The Debugger can be configured to suit your personal style and that configuration can be saved with the application. There are a number of options in the Options menu you can toggle.

Disable Debugger At Errors allows the "normal" action to occur when an error is encountered instead of popping up the Debugger window. *Disable Debugger Procedure Commands* allows you to leave your debugging commands intact, but have them ignored. *Disable Stop Key And Button* eliminates the *Stop* facility, but still allows breakpoints to pause a procedure. This option will speed execution of procedures in design mode since OMNIS doesn't have to monitor events to see if the key or the button have been used.

continued on page 33

eval() Functions

Two of the more fascinating new functions in OMNIS 7 are the *eval()* and *evalf()* functions. They are offered to us to provide both extra flexibility and, with an in-depth knowledge of their use, increased programming power. I am thoroughly impressed by this set of functions and have also found them to be extremely useful in exploring the inner workings of OMNIS 7. I hope to make you as enthusiastic as I am about their potential!

Strings as Calculations

The basic purpose of both of these functions is to convert a string to an expression and to evaluate that expression. (For a further discussion of how OMNIS treats strings and expressions, refer to the article *Strings and Expressions* in OmniScience Volume I Number 2, pages 18-21.) The differences between the two are as follows:

The basic evaluation function is *eval()*. It takes a single parameter, which can be either an explicit string or a string expression (a field name or something more complex). If the parameter is other than an explicit string, the expression is first evaluated to a string value. The string value of the parameter is then interpreted as an expression and, assuming that the expression is valid, the result of that expression is returned. The data type of the returned value is that which would normally be expected, e.g., if the original string were 'dat(#D+5)', the *eval()* function would return a date value.

The *evalf()* function is even more powerful, but is also more restrictive. It only accepts a field name as a parameter. That field must be of

a string type (character or national) to not generate an error. Here's why: When the *evalf()* function has converted the string value held in that field, it replaces the original string value with the tokenized version of the converted expression. This is done to enhance execution speed if the same string field value needs to be evaluated again (in a *Repeat* loop, perhaps). An indicator is also included in the replaced string so that OMNIS will know whether the field contains tokens or needs to be reconverted.

Testing for Validity

Since an invalid expression will halt processing with a devastating error, it is good practice to trap for invalid expressions in a procedure before using *eval()* or *evalf()*. The *Test for valid calculation* command is provided for this purpose. If an expression is not valid, this command simply sets the Flag to "false" instead of generating an error. We can then abort the process before our users are faced with their windows disappearing or the Debugger window coming to the top. Here is how we would use this command.

```
Test for valid calculation
      {evalf(field)}
If flag false
      ;perhaps an OK message here
      ;or a breakpoint if debugging
      Quit procedure
End if
```

The command can also be used with the *eval()* function. It can, in fact, be used with any expression, but it is most useful for checking these functions as shown above.

Now let's examine some of the uses we can make of this technology.

User-defined Calcs

Suppose that we are creating an application for a business that requires the ability for a management person to redefine the aging calculation or some other complex expression. We could include a string field in a System Constants File to hold the latest version of that expression and use one of the *eval()* functions with that string whenever the calculation needed to be carried out.

OMNIS has not yet evolved to the point where we also have an "Interpret" function that can turn the tokenized version of an expression (generated by the *evalf()* function) back into a recognizeable string. Since this is the case, and since the object of the exercise is to allow the expression to be viewed and changed if necessary, we should store the string in this example as entered by the operator and not use *evalf()*.

It is not a fault of OMNIS that an inverse function of *evalf()* is not yet available to us. In Omnis 3 Plus we first encountered the *fld()* function and it didn't have an inverse counterpart. In version 3.3, we were given the *Use fld() of name* option to the *Calculate* command. The folks at Blyth can only give us so much at once — and this is really great stuff!

On the data entry window where we allow the operator to insert and edit this string value, a field procedure can check for a valid expression as the cursor leaves the field. In this way, we can give immediate feedback to the operator on whether the expression is syntactically correct. We can't give them the same advice that OMNIS 7 gives us with regard to *where* the syntax error occurred or *what* the syntax error

entailed (because the *eval()* functions don't give us such a return value), but we can at least make sure that the string we are about to store is a valid expression. This, too, is still evolving, but we can take advantage of what we have right now!

Since the string can be changed by the operator from time to time, it would be a good idea to store the current value of the calculation string with the transaction record that uses it. In this way, the data will retain consistency over time.

Another point to consider in this scenario is that the field names used in the calculation string could change, rendering the string invalid. As we will see later in this article, if the *evalf()* version of the expression were stored, the expression would be independent of the name assigned to any field and would remain correct as long as the field *number* (in File Format field number order) required by the expression did not change. Perhaps this would lead us to store both versions of the expression?

Redrawing Lists

On page 11-22 of the *Application Designers' Handbook*, an excellent example of redefining column order in a List display is given. To expand that explanation a bit, we could give the operator a means of selecting the order of the columns to be displayed in a List-type window field, generate a string for a valid *jst()* function setting the proper column widths for those fields, and use either *eval()* or *evalf()* of that string as the calculation for the List-type display field.

Having done that, we could also redo the column headings for that List-type field. We would have cre-

ated a string field in a Memory-only File named COLUMN_HEAD for holding a *jst()* function string for the headings that correspond to the fields above.

A no-name display field in the same font as the List-type window field, and given the calculation of *eval(COLUMN_HEAD)*, would serve as a user-modifiable heading for the List-type field.

Since the column headings are explicit strings, the construction of the value for COLUMN_HEAD is a little more elaborate than that for the List display calculation string. We have to include single quotes around each column name.

To do this, we must use the *con()* function and *chr(39)* (the single quote character). Following the example in the manual, the heading string value would be generated as follows.

```
Calculate COLUMN_HEAD as
con('jst(',chr(39),'Name',
chr(39),',23,',chr(39),'Town',
chr(39),',8')
```

Both the List-type display field with its new column order and the no-name string field used as the heading must be redrawn after these calculations are performed.

Redefining Reports

In a similar manner, we can redefine the columns of a columnar report based on user input. One field can be used to hold a string that defines the order and width of columns, another to hold the corresponding headings for those columns along with the same widths. The only requirements of the fields in the report body are that they be assigned the same monospaced font and size.

If the report requires a Totals Section, a field for formatting a no-name field in that section to properly align totalled value under their corresponding columns can be arranged. It just takes a little more work, since not all columns will have totals. In that case, the widths of columns not bearing totals and the justification of those that do must be taken into account.

Ad Hoc Searches

We can even allow the operator to specify search criteria in a string field. This string can then be used in the *eval()* or *evalf()* function in a *Set search as calculation* command or in a calculated criterion line in a Search Format. Of course, this presupposes that the operator understands the search criteria being specified — and the proper syntax for OMNIS 7 expressions.

A calculated search cannot be optimized (forced along a specific index) in the current version of OMNIS 7, so this may not be a method you want to use in selecting a small subset of records from a very large File. On the other hand, this is the appropriate type of search to employ when searching a List. Either way, we have another useful technique to add to our OMNIS tool kit.

Tokens

Since the *evalf()* function replaces the original expression stored in a string field with the actual tokens that represent the field names, operators and functions in that expression, I took the liberty of doing a little "reverse engineering" to further explore how OMNIS works. Specifically, I wanted to see how OMNIS tokenizes the components of an expression to be able to make better decisions in the future about streamlining the expressions in my

fields per File Format). The first File created in or copied to the application is number 129, the next is number 130, etc. When you consider that there is a “low byte” (0-127) and a “high byte” (128-255) part of a byte, number 129 is equivalent to number 1 on the high byte side. We are allowed up to 99 File Formats within an application, so the range of File Format numbers is 129 to 227.

File Format number 228 indicates the system fields (hash variables). The field numbers of these fields are shown in Figure 7. Notice that there are some large gaps in the field numbers in this list. OMNIS still has plenty of room to grow within its current configuration!

Format variables are considered by OMNIS to be in File number 254. Local variables (including Parameters) are in File number 255. The field number within those pseudo-Files is the order in which they were added to the list — the order in which they appear in the Field Names window.

Possibilities

Knowing these tokens opens the door to a number of other possibilities. The mind boggles at the potential! Here are a few of the ideas I have come up with so far.

Sometimes one freedom OMNIS gives us limits our freedom in other areas. For example, we are allowed 15 characters for field names and 255 characters in expressions. The longer our field names, the fewer of them we can use.

The *evalf()* function gives us an opportunity to shorten long expressions (or pack more into them) once we understand tokens. Fields that have 15 character names can be packed into two characters! Functions that require from four to seven characters to express longhand only require one! We don't gain anything on operators, parentheses, or literals, but we don't lose anything either! If OMNIS can evaluate a *tokenized* string of up to 255 characters, we can do a lot more in expressions than first supposed!!

We can learn to minimize OMNIS evaluation time for an expression by checking the number of tokens needed to express different forms of it. In understanding tokens and the evaluation hierarchy of operators, we can quickly learn to minimize our use of excess parentheses and other common errors.

We can combine the use of *evalf()* with some other powerful functions. I envision the possibility of needing different expressions for performing a calculation under different circumstances. Suppose that we wanted to calculate an aging value slightly differently for 30, 60 and 90 day accounts. We have three fields named CALC30, CALC60 and CALC90 in our Constants File that contain tokenized aging expressions. If we have a field named TERM to hold the 30, 60, or 90, we can derive our value from the proper expression using this one line.

```
Calculate VALUE as
    evalf(fld('CALC',TERM))
```

Are we having fun yet?!

1-60	1...	STAB	76	T	100	CT	118
numerics	60	OK	77	F	101	TOP	119
P	61	CANCEL	78	L	102	LSEL	120
R	62	CLOSE	79	LM	103	FDT	121
D	63	WCLICK	80	LN	104	EFLD	122
S1	64	EDATA	81	EM	105	SUBFLD	123
S2	65	TOTOP	82	ER	106	ERRCODE	124
S3	66	TOTOP1	83	EF	107	ERRTEXT	125
S4	67	TOTOP2	84	EN	108	L1	231
S5	68	SENT	85	FD	109	L2	232
BEFORE	69	ENTER	86	FT	110	L3	233
BEFORE1	70	RETURN	87	MU	111	L4	234
BEFORE2	71	SHIFT	88	UL	112	L5	235
AFTER	72	COMMAND	89	PI	113	L6	236
CLICK	73	CTRL	90	FDP	114	L7	237
DCLICK	74	OPTION	91	RAD	115	L8	238
TAB	75	ALT	92	RATE	116	???	240
		MODIFIED	93	CLIST	117		

Figure 7 This is a list of the field numbers of the hash (#) variables within their File (File number 228).

jst() Techniques

The *jst()* function gives us incredible control over the form and content of calculated, displayed, and printed values in OMNIS. The *Programmers' Reference* manual gives a thorough accounting of the reserved characters used with this function and some examples. This article is intended to expand upon these and consider more complex possibilities. First, a brief overview.

Justification

As the name implies, the original purpose of this function, when it was introduced in Omnis Quartz, was to justify string values to the right or left on a field of character positions. The remaining positions were then filled with either trailing or leading spaces. The function required two parameters: the string to be justified and the number of characters on which that string was to be imposed. If the number was positive, left justification was used — a negative number forced right justification.

The *jst()* function will actually accept any data type for its first parameter, but it will treat that value as a string unless otherwise directed. The original function had no such option.

String Formatting

As OMNIS has evolved, more functionality has been given to *jst()* — so much so that it could better be called the “formatting” function than the “justification” function today. A few of its options are still useful for formatting strings, but there are many other options now that give us impressive control over numbers, dates and times. These tools can save you time and effort.

Composite Indexes

The *jst()* function lets us extend the functionality of OMNIS. For example, OMNIS 7 does not currently have a facility for creating composite string indexes or case-insensitive indexes. With a little extra overhead in our datafile (because of the extra storage space required), we can generate such indexes

Suppose that we would like to create a composite Last Name/First Name index in a Customer File. We would need to create an additional field in the File Format for storing this value and index it. The length of this string field should be the sum of the Last Name and First Name maximum lengths.

A composite index of this nature does not simply store the concatenation of the field values for each record. Instead, the Last Name value must be padded out to its maximum length before concatenating the First Name value. The effect is that of columnarizing the two field values represented. Forcing both values to upper case removes case sensitivity — and it doesn't effect the display of the original field values because this field is used strictly in the background with *Find*, *Next*, etc.

If the maximum lengths of these fields are 18 and 15 characters respectively, the calculation for generating our composite field value is as follows.

```
Calculate LAST_FIRST as
    jst(LAST, '18U', FIRST, '15U')
```

The *jst()* function pads each value with trailing spaces as necessary to the specified width.

We would generate the value to be stored in this field after the *Enter data* cycle and before the *Update files* in both our Insert and our Edit commands. We would have to get fancier with our *Find* procedure to use the index, but this is manageable using *Enter data* instead of *Prompted find*.

Date/Time Formatting

Just as OMNIS 7 itself allows us various date “styles” for File Format date fields, we can offer a similar feature in our finished applications. Suppose we want to allow our users the flexibility of determining the format to be used in reports for heading dates, dates of transactions, etc. We can include fields for these date format strings in either a System Constants or a User Preferences File and make available a window for the operator to modify these values. All they need to know are the date code values — and the “Field names” window does a great job of presenting those.

In our reports, we can then format the date using the *jst()* function with the appropriate field name as the formatting parameter. For example, if we use a field named HEADING_FORMAT and the operator had placed the string 'D:w, n d, y' in that field, we could format the current date in the report heading using the expression

```
jst(#D, HEADING_FORMAT)
```

and the date would be in the form “Monday, January 6th, 1992” in the printed report.

Excel Exports

A commonly used spreadsheet program in both the Macintosh and the Windows environments is Mi-

Microsoft Excel. This product can successfully import data in the tab-delimited, SYLK, and Lotus WKS formats, but sometimes the reports we would like to send to Excel require fancier formatting. For example, reports exported in the above formats don't include Subtotal Heading, Subtotal, or Totals Section values — they just send Record Sections.

We can use the *jst()* function to insert tabs (*chr(9)*) between field values for columnarization and print the report to a file on disk, to the clipboard, or to Publish on the Macintosh. We can do this in *any* section of a report, making sure our subtotals and totals are in the proper column.

We can also use this function to convert dates to the proper format for Excel. That program expects to see dates in a similar format to the one that OMNIS uses, but the month abbreviations are *capitalized* instead of all upper case. To format dates for proper import into Excel, use the following expression. For European date formats, simply transpose the month and day parts.

```
jst(date value , 'DC:m D Y')
```

List Columns

This is one of the most common uses of the *jst()* function. The function provides a means of columnarizing multi-valued strings for display in a List-type window field. More rarely, it can be used for the same purpose with Popup Lists and Combo boxes. For most field types, the alignment we get is perfect — but some date formats require a little more attention.

European dates always line up nicely in Lists and reports by sim-

ply right justifying the field value. But when generating dates in the American format, with the month abbreviation before the day number, date values treated in the simplest way don't line up nicely in columns in List-type window fields or in reports. If the date value is left justified, the years don't align — and if the date value is right justified, the month abbreviations are out of alignment. A List display or a report can look much neater if these columns *both* line up and if the day number is right justified.

All we need to do is split the date into three columns — one for each of the parts of the date value. *jst()* is very handy for this, since it implies concatenation and has those wonderful reserved characters for date formatting. The expression we would use looks like this.

```
jst (DATE, 'D:m', DATE, '-3D:D',  
DATE, '-3D:Y')
```

Note that we didn't have to specify the number of characters for the month abbreviation, since it always has exactly three characters. We specified three characters for the day part and right justified the column, which leaves at least one space between the month abbreviation and the day number.

The same reasoning applies to the year number. In each case, we had to indicate that we wanted to use the field value as a date by using the letter "D". The date formatting reserved characters would not operate correctly without applying this "type" reserved character first.

If these columns are to be followed by a left justified column, a "gutter" should be included between the two groups. This is done by including two more parameters to the *jst()* list. These are one or more

explicit space characters (spaces between single quotes) and a null string (two single quotes with nothing between them) indicating no special formatting. The null string is necessary since the *jst()* function does not allow two adjacent delimiting commas.

COBOL-style Reports

A question that comes up often in classes when we discuss exporting data to other applications is how to generate fixed-length, space-padded output strings so that data can be passed to applications (often COBOL-based) that can only import in that format. The solution again involves the *jst()* function.

The Report Format for this reporting style will have only the Record and End of Report Sections. It will also have a single no-name field or a single text block using square bracket notation. The expression used in either case will be a *jst()* statement that properly columnizes the fields to be exported.

The Report Format can be left as a standard report, that is, we don't need to specify an export format. Simply printing the report to a file on disk performs the proper export. Some of the export formats would be completely inappropriate, while others would introduce unwanted double quote characters that we would have to strip out.

String values should be left justified and numbers should be right justified and assigned the appropriate number of decimal places. Dates and times are generally stored as strings in databases that require this type of import, but you should check on specific formatting requirements for the target database. *jst()* can handle any special requirement with ease!

Ad Hoc Reports

OMNIS programmers have long awaited a facility where the end user could define reports as they are needed — *without* the end user needing significant knowledge about OMNIS. Although this is impossible in the strictest sense, the desire for a feature like this has validity. OMNIS 7 has taken a great step forward in providing us with such a facility to offer our users.

The term “ad hoc report” connotes the ability to create a complete report from thin air to many people. This is not at all what the Ad Hoc Report feature of OMNIS 7 offers. It *does* offer the end user the ability to determine the list of fields to include in a report, but the report is generated within the confines of the “template” the user chooses. The template determines the structure of the final report. If a template with the desired structure has not been provided by the programmer, an Ad Hoc Report of that structure cannot be built by the operator.

This is by no means a complaint about, or problem with, this great feature. These are simply the ground rules — like the statement, “the operator can’t store data without the programmer first creating a File Format and a data entry window.” It is the responsibility of the OMNIS programmer to provide for the needs of the end user regarding templates.

Reports Menu

This is the gateway to the Ad Hoc Report facility in OMNIS 7. This menu (Figure 8) can be installed in a number of different ways. Your first access to it is likely to be through the Menus submenu of the

standard File menu. Selecting the *Reports Menu* option installs the Reports menu on the menu bar. It can also be installed by executing the *Install menu* command. Sometimes, in a finished application, it may be more appropriate to install it as a hierarchical menu from a more general, user-defined Reports menu using the *Install hierarchical menu* command (my personal preference).

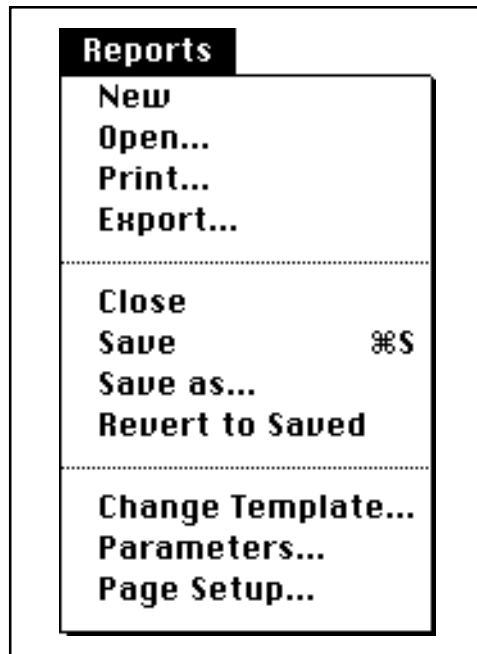


Figure 8 Reports menu

If an Ad Hoc Report is not currently open, only the first four options, *New*, *Open*, *Print*, and *Export*, are available — the rest are greyed out. *New* allows the operator to create a new report based on an existing template, the others allow the operator to perform operations using existing reports that have already created and been saved to disk. We will discuss saving and using Ad Hoc Reports later in this article.

The commands in the other sections of the Reports menu apply only to the currently open Ad Hoc Report.

Using a Template

Selecting the *New* option from the Reports menu brings up the dialog box shown in Figure 9. The operator is prompted to select a template from those presented — and can go no further until this is done. The only other option is to Cancel.

The templates are actually Report Formats stored in a special application named Report Templates. A Report Templates application is supplied with your copy of OMNIS 7 that contains a starter set of three basic Formats: a columnar report, a paged record or dossier report, and a mailing labels report. These can be modified to suit your particular needs and more templates can be added to this basic set. (See the section on “Creating Templates”.)

Once the operator has selected a template, OMNIS then prompts for the File(s) and fields to be used in the report. The fields are selected from the list presented in the standard way (by clicking, dragging, etc.). Once the fields have been selected, the operator clicks the “Finish” button and is presented the Fields window in the Query mode.

Field Views

There are three views of the Fields window. The Query view is the most often used. This is the view that allows us to specify most field attributes and search criteria for the report (yes, setting search criteria is a *built in* feature of Ad Hoc Reports!). The other views are the Titles view and the Calculations

view. Since these are more limited in scope, let's examine them first.

By default, OMNIS 7 will label fields or columns with the name of the corresponding field. Since these are all upper case and may have underscores in them, they don't make for a "finished" presentation. The Titles view allows the operator to assign an alternate title for each field. This title will automatically appear wherever the template dictates — and that text block can be moved by the operator if necessary.

The Titles view also allows the entry of formatting attributes for the selected field. The Format submenu is shown in Figure 10. It displays various numeric and date/time formats in a manner similar to that used by Microsoft Excel. The formats that it assigns, however, are those used by the *jst()* function. These can be overridden and enhanced once they have been assigned.

The format for a field appears after the title and a back-slash ("\") character. The formats specified can be modified in the text area at the top of the window. There are many options. For example, a number after this back-slash limits the field value displayed on the report to that number of characters. Any of the formatting attributes recognized by the *jst()* function can be used here.

The Calculations view allows the operator to change the expressions assigned to Calculated fields. It is important to only attempt to modify Calculated fields in this view. The placement of Calculated field is covered a little later in this article.

The Query view allows the operator to manipulate three basic things about the report: the attributes of the fields (through the Field menu), the sort order of the records in the report (through the Sort menu), and the records that will be included in the report (through the Query menu).

The basic field attributes are controlled by the bottom set of options in the Field menu. Here we can toggle the *Invisible* and *Totaled*

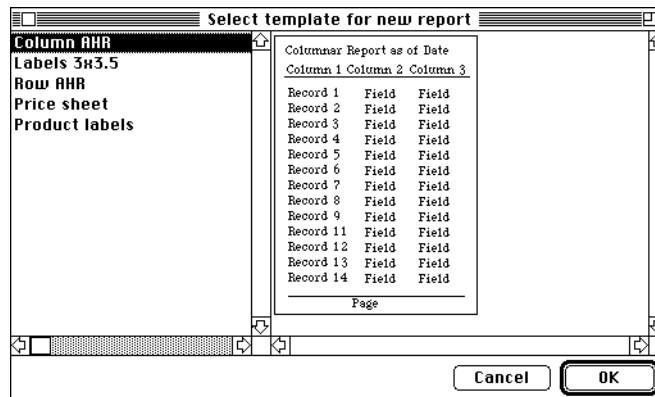


Figure 9 Select template window

attributes and assign special formatting to a field.

Another attribute that can be changed is the field name to be assigned to that report slot. As with window fields, a field on a report is just an area that has attributes, among them may be the name of a field whose value is to be displayed. The *Alter Name...* option in the Field menu allows the operator to assign a different field name to the selected report field. This *does not* alter the name of the corresponding File Format field, it just reassigns a *different* File Format field.

All the above Field menu options apply to the currently selected field. That field is selected by clicking on the line containing that field's in-

formation, among other methods. It is also possible in Ad Hoc Reports to select an insertion point *between* two lines (fields). If you move the mouse slowly over the field list vertically, you will notice that it changes shape when it is over the "divider" between two fields. If you click the mouse when the pointer has this shape, a dotted line will appear between the two lines. It is on such a dotted line that the other options of the Field menu operate. If this insertion point has not been selected, these menu options will make their contributions at the bottom of the field list.

For example, the *Add Fields...* option displays the Add Fields dialog box and any fields selected are added to the field list at the insertion point. *Add Calculation* allows the operator to create one or more Calculated fields. These are assigned to their own File Format in a similar way to Format Variables in Window Formats.

Queries

The fields on this window are dealt with by the querying system in the order they are presented on the window. For simple queries, this order won't really matter, but for complex, hierarchical queries involving multiple levels of AND and OR, it matters a great deal. The *Add AND* and *Add OR* options on the Field menu are used to insert these Boolean querying operators at the insertion point. These are used to combine query attributes assigned to different lines.

Each field can be assigned its own query attributes. The Field List window includes a palette of query

operators that can be used to build query criteria for the selected field. These criteria appear in a text entry area at the top of the window as they are selected. The criteria can be modified from the keyboard or even entered directly, if desired. The syntax of a query line here is similar to that of OMNIS Boolean expressions, but there are many more options.

For example, a single line can contain multiple query criteria. This works in a similar way to lines in a Search Format. The separate criteria are delimited by commas. Here is an example.

=A, =B

This query is assigned to a string field. There is an implied OR between the two queries, since it is impossible for the field to be both equal to "A" and "B" simultaneously. Most other combinations would have an implied AND between them.

Since the query is for a string field, quotes are not required around the literal string values. Numeric fields can contain calculated criteria, but literal strings in these expressions must be delimited by single quotes.

The majority of the query operators are comparison operators, but there is also a tool in the palette that pops up a menu of pattern matching, null filter and relationship query operators. The pattern matching features are very impressive, as is the ability to specify *Null* or *Not Null* values. The pattern matching operators *Like* and *Not Like* accept single character and any characters wildcards.

The single character wildcard is "%". The any characters wildcard

is "_". Either or both of these can be embedded in the pattern matching string multiple times. The trick is that only values that match the *entire* string are accepted. Suppose we want all records where the selected field ends with the string "ing", our criterion would be

like _ing

If instead we want records where this field contains "ing" in the

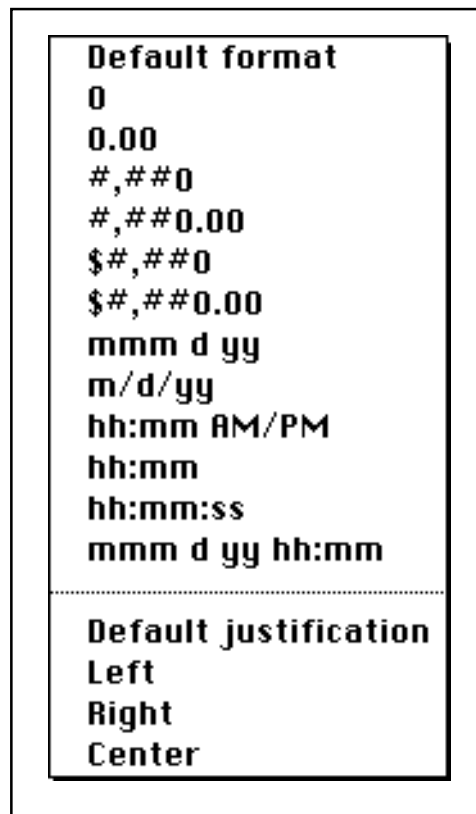


Figure 10 Format submenu of Field menu

middle, but neither at the beginning nor at the end, our criterion would be

like _ing_

Values with "ing" at the end would fail this test because OMNIS expects something *after* the "ing" — unless, of course, "ing" is in the string more than once.

Sort Fields

A field is made a Sort Field here by selecting it and using the Sort menu. Each new field that is added to the list of Sort Fields is added either at the top or at the bottom. These are the meanings of the *Sort First* and *Sort Last* options from that menu. A field can be removed from the list by using the *No Sort* option on it. The Sort Fields are distinguished in the list by icons that indicate their sort level and an arrow that indicates an ascending or a descending sort will be performed on that field.

Once a field has been made a Sort Field, the standard Sort Field attributes can be applied, if needed. These are the *Descending*, *Upper Case*, *New Page*, and *Subtotals* options. Selecting the *Descending* option changes the arrow icon to pointing downward. Selecting any of the other options adds an additional icon for each — a "U", an "N", or an "S" respectively. These icons all appear between the Field and Query columns.

There is also a *Clear All Sorts* option in the Sort menu that resets the Ad Hoc Report to having no sort fields.

Totals and Subtotals

If Totals or Subtotal Sections were added to the original template, the operator can specify fields for generating subtotal and total values. Every field marked as *Totaled* will display a cumulative value in the Subtotal and Totals Sections.

As in Report Formats, a number of things have to come together here for totals to print. First, there have to be Totals and/or Subtotal Sections. Next, the operator must specify the proper Sort Fields as

being *Subtotaled*. Finally, and here is where the difference lies, some field somewhere in the report must be marked as *Totaled*.

That's right. If no field is being totaled, no Subtotal or Totals Section prints. Even if you only need to have a new Subtotal Heading printed at the beginning of the next subtotal grouping, some field must be totaled. If you don't need to numerically total anything, but would still like to trigger a subtotal break for formatting purposes, here is what you should do.

Make a calculated field that is calculated as zero. Give it the "zero shown empty" attribute with the formatting features "NE", and make it *Totaled*. Your subtotal and page breaks will now be triggered.

It is rare that a correction to the wording in the OMNIS manuals is needed, but one passage at the bottom of page 8-13 of the *Application Designers' Handbook* needs clarification. If a field is marked as being *Totaled*, an appropriate value (total or subtotal) for that field will appear in *all* Subtotal Sections as well as the Totals Section if that field name is referenced there. A Sort Field that is given the *Subtotals* attribute *will not* generate subtotal or total values unless it is also marked as *Totaled*. A change in the value of a Sort Field that has this attribute simply *triggers the printing of the corresponding Subtotal Section* and, therefore, of the subtotal values for that subtotal level of all *Totaled* fields.

It is *extremely rare* that a Sort Field will also be *Totaled*. Most often we would want to trigger subtotals on some numeric values based on a change in a string field. For example, we might want a subtotal of invoice amounts based on a

change in customer name or ID number. A subtotal of all the ID numbers in the last (exact match) group would be meaningless — and a subtotal of all the names in the last (exact match) group would be zero since strings have a zero numeric value.

For further information on subtotals, and other reporting issues, refer to "The Omnis Report Generator" in *OmnisScience Volume I Number 1*, pp. 26-29.

Customize Window

The Ad Hoc Report navigation palette is in the upper left corner of every Ad Hoc Report window. The Field List window is entered by clicking the icon on the far left that has horizontal lines on it (looks like a list). The Customize window is entered by clicking on the next button — the one that has the image of a pencil on it. This is where the layout of fields on the report is manipulated and it is the next window our operator normally needs to work with.

The fields chosen for the report on the first window appear in that same order here, but in the format dictated by the template selected. They do not need to remain in the same order. The operator can rearrange them in any order that seems appropriate. Columns can be switched, and even new lines can be added, on columnar reports. Label fields can be shuffled. The order of fields on the Field List window is for querying purposes, not output format. The *form* of the output is set on this window.

The operator can also add any graphic elements and text he or she deems necessary. Any features that the operator *removes* that were automatically generated, however,

will not be automatically *replaced*. These are things like column headings, fields in the Totals Section, etc. The operator does not have the same freedom in this environment that we are used to in the Report Format modification window — but neither does the operator have the same responsibilities or require the same knowledge. The idea was to make the process simple.

This window also contains a tool palette and a palette with fill patterns and line types. If color is available on the operators computer, Text and Background color palettes also appear. The color styles can be customized and they are stored with the *report*, not with the application like the color table accessed through the Defaults menu. Each report can have its own custom color set! The same goes for the Fonts menu.

Trial Output

As the operator works at enhancing the report, there are two buttons on the Navigation palette that can be used to show the report in its current form. These are the Output and Rebuild buttons. They have icons that look like a report and a report with a check mark on it respectively.

The Output button should only be used once per session. It builds an initial report based on the current sort, query and format criteria and displays it on the Output window. The Rebuild button is used to update that report as changes are made later in the session.

At any time, the current version of the report can be printed to the printer or to a file on disk using the other Navigation buttons. Printing can also be done from the "P" menu on the Output window.

Saving Ad Hoc Reports

Once an Ad Hoc Report has been modified to the operator's liking, it can be saved to disk for later retrieval and use. The operator can choose either the *Save* or *Save as...* options from the Reports menu. If the report has not yet been saved, these options operate identically and prompt the operator for the name and directory location to be assigned to the report file.

If the report has already been saved, but has since been modified, the *Save* option (command-S) will save the current version of the report to the current pathname. The *Save as...* option will save the current version to a new location and name, specified by the operator, retaining the old version on disk. In this way, the operator can make variations of an already satisfactory report and save them separately.

Page Setup

One of the great advantages to Ad Hoc Reports is that they are *stored with their page setups*. Because of interface guideline restrictions, we are not allowed to store page or job setup information with standard Report Formats. This is because they are viewed as only *part* of the document — the “document” being the OMNIS application. Since Ad Hoc Reports are stored separately from any application, they are considered separate documents and can contain custom page setup information.

You may find that you prefer to make many of the reports for your applications pre-defined Ad Hoc Reports to take advantage of this feature. If you ever need landscape orientation, size reduction or special print options selected for a report, this is a viable option.

To access the page setup information for an Ad Hoc Report, the operator only needs to select the Page Setup option from the Reports menu. The standard page setup dialog for that computer type and the selected printer will appear.

Printing

An Ad Hoc Report can be printed in any of three ways. Any time that the report is open, it's current form can be printed by clicking on the Print icon. The *Print Report* option from the “P” popup menu no the Output window performs the same function.

If the report is not open, the operator can print it in its stored form by selecting the *Print...* option from the Reports menu and then selecting the name of the Ad Hoc report file desired.

The third method uses a procedure. The *Print ad hoc report* command operates in nearly the same way as the *Print...* menu option. The difference is that the procedure command must be supplied with the pathname of the report to be printed without direct intervention on the part of the operator. The only problem with this is that the pathname *must* be known for the operation to be successful.

Another method of procedurally printing an Ad Hoc Report is to call the *Print...* option from the Reports menu as a subroutine.

Creating Templates

Existing templates can be modified and new ones can be created by the OMNIS programmer. It is simply a matter of opening the Report Templates application and working with Report Formats there under a few guidelines. The Report

Templates application should be located in the same folder or directory as your copy of OMNIS 7.

When you open the application, the only Formats of significance that you will find are the Report Formats that make up the set of templates and a single Memory Only File named “AHF”. This File contains a single field named “AHF_NAME” that is given the current name of the Ad Hoc Report that you are using. If the report has been stored, it is the name of the file on disk. If the report has not yet been stored, the name will have the form “Reportx” where “x” is a number indicating the number of Ad Hoc Reports you have created in this session.

The AHF_NAME field can be placed in any template so that the stored name of the report can be printed. The AHF File will appear in the list of File Formats while working with an Ad Hoc Report in any application.

With regard to the fields that will appear in a finished Ad Hoc Report, templates are completely defined using Report Format fields for #1 and #2. The relative positions of these two fields determines whether the Record Section of the report will have a vertical or horizontal orientation. If the fields are placed side by side, the report will be columnar. If they are placed one above the other, a dossier or label layout is the result.

Column headings or field labels are assigned by using text blocks, but we don't want to give hard-coded values to these. Instead, we use the strings “#1” and “#2” to indicate where the headings for the corresponding fields should go. In the Ad Hoc Report itself, the *name* of the field will be supplied

by default in place of these indicators, but the field names can be overridden by the operator by specifying “titles” for the fields. The assigned title will then replace the “#x” place holder.

Other characters can be included in these strings as well. Only the “#x” will be replaced. This allows us to create column headings that look like “ —•heading•— ” if we like. Commonly used characters for field labels in vertical format reports are colons, dashes, greater than symbols (used as arrows), and equal signs.

Fields or text blocks referencing various other system variables may also be included in a report template. Frequently used variables are: #P (page number), #R (record number), #D (current date), #T (current time), and #SUBFLD (equivalent to or alias for the field triggering the current subtotal level). For “live” values of these fields to appear in text blocks, square bracket notation must be used. Temporary fields used for calculations are left to the user of the template and are not appropriate here.

Graphics and descriptive text blocks can also be added to the template. Commonly used graphics are company logos and lines used as column, record or section separators. Column separators only need to be placed between the #1 and #2 fields in the Record Section. When fields are added to the actual Ad Hoc Report, OMNIS will insert the same graphic between *each* pair of fields automatically.

The sections used in a template are the sections that will appear in any finished report using that template. The operator has no access to section attributes or to creating new sections. The operator can only

move or delete section banners in the Ad Hoc Report.

If a finished report is to contain subtotals, the *appropriate* Subtotal Section(s) must be included in the template. If the report is to have a single subtotal level, but the subtotalling will be based on Sort Field number 2, Subtotal level number 2 must be used in the template. If two levels of subtotalling are required in the finished report, and Sort Fields 3 and 4 are to trigger those subtotals, then Subtotal levels 3 and 4 must be the Subtotal Sections used. The operator then needs to be informed of these facts. The best way of informing the operator is in the name of the template.

The name of the template that the operator sees when selecting a template for a new Ad Hoc Report is the description the programmer gives to the Report Format in the Report Templates application. This allows template names that are not restricted to fifteen characters and that contain characters that might prove confusing if used in a Report Format name.

For a columnar report that is to be totaled on Sort Fields 3 and 4, we might assign a name like “Columnar Subs 3,4”. Our operators would have to be trained to recognize what this means, but having these options in an application of real world complexity gives us enhanced data management power!

A template preview can also be provided by the programmer to help the operator visualize the form that a report would take for a given template. The programmer must create this preview in a drawing program that allows for the proper type of graphic output. On the Macintosh, PICT output is required

— in Windows, PCX output is required. The graphic should be detailed enough to get the point across, but simple enough to not cause additional confusion.

The finished template preview is then pasted onto the template view of the Report Format on a line *before* the first section banner. It must be the *first* item in the Report Format (reading from top to bottom). The Paste can be done from the Clipboard, the Scrapbook, or by using the *Paste from file...* command. It is not necessary that the *entire* graphic appear before the first section banner, merely that it *belong* to a line with that characteristic. Remember that any object in an OMNIS 7 report belongs to the line on which its topmost row of pixels belongs.

Opportunities

It occurs to me that there might be an opportunity here for people who are really good at creating nice-looking reports or precision forms to make a little income. Since the Report Formats in the Report Templates application are completely independent of any Files, they can be used by *any* application. I can envision really hot Ad Hoc Report Templates being bought and sold or traded among friends and associates. There are all the commercially available laser printer and pin-feed labels to write for, for starters! Commercial templates only need to be moved into the Report Templates application with Application Utilities.

Finished Ad Hoc Reports can also be swapped, but on a more limited basis, since they rely on their associated File Formats. Still, within a work group, there are probably plenty of opportunities for the Report Wizard.

Tips and Techniques

A Matter of Degrees

The trigonometric functions in OMNIS work successfully with either degree or radian input depending on the setting of the #RAD variable. But, when working in degrees, OMNIS needs to see a *decimal* value.

Often it is necessary to display angular measurements in degree-minute-second format (DMS). For example, longitude and latitude are generally expressed in this way. If some of your OMNIS applications need to display DMS degrees, or if you ever need to import information from a source in this format, the conversion procedures in this article will come in very handy.

Let's first consider converting decimal degrees to DMS format. For this example, we will use #1 to hold the decimal degree value. The degree portion of our output can easily be seen as the integer part of the decimal input or *int(#1)*. We can determine the number of minutes in the fractional part by converting the value to minutes (multiplying by 60), removing the degrees portion (taking the modulus of the number of minutes by 60 minutes), and taking the integer part of the result (since our modulus value may have a fractional part).

The same process is used to derive the seconds portion, although we use a factor of 3600. We don't take the integer of the result since we would like to see any remaining fractional value in the seconds portion. It is then a simple matter to introduce the proper symbols to delimit the three values in our resulting string. Here is a simple procedure that displays the result.

dec to dms conv

```
Calculate #1D4 as decimal value
OK message (High position)
  {[#1] degrees is [int(#1)] °
    [int(mod(#1*60,60))] '
    [mod(#1*3600,60)] '' }
```

If we wanted to generate the value for storage, we could have calculated the string using the *con()* function (see below).

Converting a DMS value to decimal degrees is a little more complex. First we have to extract the numeric degree, minute, and second parts from the string. The math from there is easy.

We can extract the numeric parts of our DMS string by knowing where the degree, minute, and second symbols are located within the string. Since we will need to use most of these position values more than once, it is easiest (and fastest) to set up temporary variables to manipulate them.

The *pos()* function is used for this purpose. (Note that we need to refer to the single quote character that indicates minutes by using the *chr()* function because it is otherwise used as a string delimiter in expressions.) The *mid()* function is then used to extract the numeric values from the DMS string using these numeric position markers as shown below. The resulting strings contain only numerals, so they can be used as numbers in the expression with no conversion. These values are then converted to a common basis (degrees) by dividing the minutes part by 60 and the seconds part by 3600. The intermediate results are ultimately added to obtain the final result. Here is an example procedure.

dms to dec conv

```
Calculate #S1 as dms value
Calculate #10D0 as pos(' °',#S1)
Calculate #11D0 as pos(chr(39),#S1)
Calculate #12D0 as pos('','',#S1)
OK message (High position)
  {[#S1] is [mid(#S1,1,#10-1)
    +mid(#S1,#10+1,#11-#10-1)/60
    +mid(#S1,#11+1,#12-#11-1)/3600]
  degrees}
```

These procedures can also be cast as “black box” modular procedures (aka user-defined functions) by using *Parameter*, *Local variable* and *Set return value* commands. The *Call procedure with return value* command would then be used to pass parameters to the procedures and retrieve the results.

dec to dms conv black box

```
Parameter {DECIMAL_VALUE}
Set return value
  {con(int(DECIMAL_VALUE),' ° ',
    int(mod(DECIMAL_VALUE*60,60)),
    chr(39),' ',
    mod(DECIMAL_VALUE*3600,60),' '' )}
```

dms to dec conv black box

```
Parameter {DMS_STRING}
Local variable {DEG_POS}
Local variable {MIN_POS}
Local variable {SEC_POS}
Calculate DEG_POS as
  pos(' °',DMS_STRING)
Calculate MIN_POS as
  pos(chr(39),DMS_STRING)
Calculate SEC_POS as
  pos('','',DMS_STRING)
Set return value
  {mid(DMS_STRING,1,DEG_POS-1)
    +mid(DMS_STRING,DEG_POS+1,
    MIN_POS-DEG_POS-1)/60
    +mid(DMS_STRING,MIN_POS+1,
    SEC_POS-MIN_POS-1)/3600}
```

These are just two of the many useful conversion routines that can be found on my “Black Boxes” example application. See the Products section of this issue for more information.

Global Documentation

One of the features high on the wish list of many veteran OMNIS programmers is the ability to document a number of Formats in a single command. It is not readily apparent, but this facility is available in OMNIS 7 if you know where to look.

Actually, this feature was a last-minute addition to OMNIS 7, so there wasn't much time to add it to the manuals.

The is feature is the *Print* button on the Application Utilities window (Figure 11). On page 12-8 of the *Application Designers' Handbook* under the title "Printing a Field List" it says, "This option prints a field list for each selected format to the current report destination." Let's explore this further.

Certainly for a selected File Format this button will print a list of the fields in that File, along with the attributes and descriptions of each field. The File Format listing also includes Connection information and an estimate of disk usage for that File.

For Window and Report Formats, it prints out much more than a simple list of fields — and for Menu and Search formats (which don't have "fields"), it prints out appropriate details. Think of this as the "Details" or "Documentation" button. For Window Formats, it prints a listing of fields *and their associated procedure and other vital attributes*. It prints virtually every

bit of information about each field, including the horizontal and vertical position of the upper lefthand corner of the field in window coordinates and the justification applied to the field.

The fields are detailed in Window Format field number order beginning with the initialization procedure (number 0) for the window. All procedures in the Window Format are listed, even if there is no associated field.

NAME	TYPE	BLOCKS
Customer file	file format	3
Donation file	file format	1
Flags	file format	1
Inventory file	file format	2
Invoice file	file format	3
Items file	file format	1
System file	file format	1
User file	file format	2
Commands	menu	1
STARTUP	menu	15

* = protected from alteration

Figure 11 The Application Utilities window now includes a "Print" button

Report Formats are detailed one section at a time from the top of the format. Each field is listed in left-to-right order a line at a time. The field attributes are given for each as well as any lookup table or calculation string. The position of each field is given by its line number *within the section* and the horizontal position from the left margin.

The output for Menu Formats is a listing of the procedures in each one in procedure number order. Search Formats list each search criterion line in line number order.

So if you've been looking for a way to document your entire application — or, at the very least, a number of Formats simultaneously — the facility is there. There just wasn't time to document it.

Debugger

continued from page 19

If *Default To Show Debug Menus* is switched on, all procedure windows will automatically include the Debugger window overlay. You may very well find this your preferred option, but it is a matter of choice. *Default To Read Only Mode* applies to procedure windows, not to File Formats. Read Only Mode disallows editing of procedure commands *and* causes tracing to proceed faster by dispensing with redraws. It can be toggled by the *Read Only Mode* option in the Debug menu. This `command` causes the application to default to this mode. (The default default mode for newly created

File Formats will continue to be Read/Write.)

Your personal combination of options can be saved by selecting *Save Debugger Options*. They are saved to a table format named #DEBUG along with the maximum number of lines you specified for the Trace Log. The next time you open this application, your combination of options will be automatically selected. If you like this mode of working, you can copy #DEBUG to other applications using Application Utilities like any other Format. If you choose to override your defaults, you can regain them again without closing and reopening the application by selecting the *Revert To Saved Options* item on the Options menu. Fantastic features, incredible convenience!

Tips and Techniques (cont.)

Default Formats

There are times when you just can't please everyone. There were already so many different versions of OMNIS 7 being created and released simultaneously that a decision was made to not differentiate American and British versions on the first release. The only things that this effects are default date formats for File Format fields and default date and number formats for Ad Hoc Reports. Thought is being given as to how to address this issue in future releases (perhaps with Installer options), but we can fix this fairly simply today.

This fix is only for Macintosh versions (currently) and should only be performed by an advanced user. The fix is also only for the copy of the product you alter — the process will have to be performed again when you receive updates until such time as the new version does not require the fix. Blyth Software cannot be held responsible if you choose to follow this advice. Poly-math cannot be held responsible if you stray from the advice given.

Here is the problem. The default date formats for File Format fields in any newly created application will be in European order (day number before the month abbreviation). Also, not only the default date formats, but the default "money" number formats in Ad Hoc Reports are British — with the £ symbol where the \$ symbol should be for US or Canadian currency.

True, we can transfer the table format #DFORMS to other applications using Utilities once we evolve a preferred set of date formats, but these are stored with

each *application*. It is more convenient if the *defaults* for *new* applications are "correct" (in American) in the first place. (And #DFORMS still doesn't address the Ad Hoc Report problem.)

Although our licenses with Blyth for OMNIS 7 preclude our reverse-engineering the product, resources in Macintosh programs are available to everyone through resource editors like ResEdit. In fact, Blyth Software has, in the past, published tech notes detailing how to use ResEdit to change resource strings for specific needs. We will simply alter the string resources used to generate the date and time format default strings. Here is the procedure to follow to solve the problem.

First, get a copy of ResEdit (version 2.1 or higher) from your local computer dealer or from any of the online information services, like CompuServe. It comes with a text file that serves as documentation.

Now start the program and open your copy of OMNIS 7. You will see a window with icons that represent the various types of resources used by the programmers of OMNIS 7. Most of these you should leave alone, but some of the string resources can be customized without affecting the way OMNIS performs. We want the STR# resources at the bottom of the window.

Double-click on the icon representing this group of resources. You will see a listing of the resource numbers. Each resource number contains one or more strings that are used within OMNIS for one thing or another. Within each resource, the strings are numbered beginning with number 1.

Our first stop is the string that gives the default value to #FDT. This is the first string (number 1) in STR# 1680. If yours has a value of "D m Y H:N:S", you can simply switch the "D" and the "m". If you prefer 12-hour AM/PM times, you could also change the "H:N:S" to "h:N:S A" while you're here. All the codes we use are the ones given in the "Date codes" selection shown in the "Field names" window in OMNIS 7's design mode.

Among the strings in STR# 1700 is the default for #FT (number 9). Again, if you prefer 12-hour times, change this to "h:N A". Also, if you prefer your AM and PM indicators to be in lower case instead of upper case, they are string numbers 19 and 20 in this resource.

But our real concern is with the default date types used for File Format fields. These are found much further down the list in STR# 17040. This resource contains seven strings. They are the default date and time formats you are initially presented in the File Format design area if you have not imported or created the #DFORMS table.

We can also add new default strings to this list. You will notice that the entry field for the seventh string is followed by a line with the characters "8) *****". Clicking on those characters allows you to use the *Insert new field(s)* command from the Resource menu. The string you insert will become a default date and time format in any application that doesn't contain #DFORMS.

The default number and date and time formats for Ad Hoc Reports present a slightly different challenge. We have two things to change for each format — the format string itself and the display of that string in the Formats submenu.

The strings that display on the Formats submenu are found in STR# 24800. The authors of this part of OMNIS 7 chose to display these formats in a manner consistent with Microsoft Excel, assuming that the end user would be familiar with this scheme. The ones we may wish to change are the monetary numbers (strings 7 and 8 in this resource), the dates (9 and 10), and the date and time combined format (14). For the monetary numbers, we want to replace the £ with a \$. For the date portion of the others, we simply need to switch the month and day parts.

The actual formatting strings generated from those menu selections are stored in STR#s 24840 and 24860. These strings are used as though they were parameters in the *jst()* function in that the data type must also be given. That is, dates all start with "D:", numbers are formatted like "Nx" (x is the number of decimal places), etc. In STR# 24840, the monetary number formats are strings 17 and 18 and the dates are strings 19 and 20. In STR# 24860, the date and time default string is number 4. Just make the equivalent modifications to those we made above.

If you want to include a custom format for a fixed number of digits numeric type with leading zeros, the Excel format for this would be "00000" for five places. The OMNIS 7 equivalent is "-5N0P0".

I must again caution you against unnecessary poking around in *any* program with ResEdit unless you know what you're doing. As a friend said the other day, "You could *really* get into trouble with this tool!" He's right. I present this tip because it solves a problem many of us face. I am *not* advocating wanton hacking in the OMNIS 7 code!

Record Locking

I noticed something the other day that reminded me of how OMNIS works. I thought I should share it with you.

Using two copies of OMNIS 7 on a single machine accessing its own hard disk, I was attempting to demonstrate multi-user record locking to a client. From each copy of OMNIS 7 I opened the same application, opened the same window, located the same record, and clicked the *Edit* button.

Prepare for edit on the same record from both copies was allowed at the same time! The second user was not locked out! I was surprised for a second — then I remembered how the process is *supposed* to work.

In OMNIS, the locking is actually done by the server (network) software, not by OMNIS. OMNIS sends a *request for lock* to the server software. Since I wasn't accessing the datafile over a network, there was no server to perform this function and both edits were allowed.

This is not the normal configuration for applications, but it does make single workstation multi-user demos impossible. The worst that this means for me is that I need to carry around a few more machines to demonstrate multi-user access.

I repeat, *this is not a problem with OMNIS*. In the real world, no one would be operating two copies on a single workstation anyway. But it's situations like this one that remind us how no software package stands alone.

OMNIS cooperates with the server software to give you safe multi-user access to data.

Custom Edit Menu

Normally when you issue a command to *Disable all menus*, the Edit menu should still be available. This is true in OMNIS, but only for the "official" Edit menu — a "custom" Edit menu installed with the *Replace standard Edit menu* command will be disabled by *Disable all menus*.

The remedy seems fairly simple. We just need to disable all the menus *except* the replacement Edit menu. Unfortunately, we don't have a specific *Disable selected menus* command — and there are other complications.

We *can* disable selected menus by disabling their zero procedures. This is a little cumbersome since we have to execute the command once for each menu mounted on the menu bar, but it *is* doable.

Since the *Disable menu line* command is reversible, we can perform that series of actions in a reversible block and have the menus automatically re-enabled no matter how the procedure ends. The problem with this approach is that we must disable each of the menus *each time we need to*. We can't call a global subroutine to do this task *and* have the effect reversible. Using a reversible block in the subroutine simply reverses the disabled state at the end of the subroutine. Calling a procedure from inside a reversible block *does not* cause that procedure to be reversed.

If we use a global subroutine to disable those menus, we also need a global subroutine to enable them again. Admittedly, this is a little fussy, but fine details of interface issues are what make GUI applications so desirable. I'm sure that OMNIS will evolve to address this.

About the Author

When the management at Blyth Software approached me about a blanket subscription to my newsletter, they suggested that I should tell you a bit about myself.

I have always felt that my position in explaining to the world why OMNIS is such a great product is a much stronger one if I remain independent, so I have never become an employee of Blyth Software. I *have* become expert at a number of database development products and I am always open to new ideas — but OMNIS 7 is by far the most sophisticated database development product I have encountered on *any* personal computer platform and it remains my platform of choice.

Through my company, Polymath Business Systems, I have been contracted by Blyth on several occasions over the last seven years to write training materials, perform competitive testing on their products for their marketing efforts, create OMNIS applications for their use, and conduct in-house training courses on OMNIS for both their clients and their employees. For much of this time, I was the primary source of training on OMNIS in the United States and I continue to offer instruction in open classes (for general students) or on-site (for corporate clients) across the nation.

I also wrote a book, *Unlocking Omnis 3 Plus*, that has been used by many people as an extension of the manual for that version of the product. Many still purchase it as an introduction to concepts in Omnis 5 and 7. One or a number of books on OMNIS 7 are *definitely* on my list of the top ten projects for the next year.

My office often receives calls that should have gone to Blyth Software (such as requests for upgrades, etc.). While I am flattered to be so linked with Blyth in the minds of these people, I can only offer certain services. To smooth the road for all concerned for this OMNIS Renaissance period we are now entering, let me clarify the differences between our two companies.

Blyth Software creates and sells OMNIS 7. Although I write extensively about that product, I do not sell it. I cannot take any credit for its fantastic features nor do I have any say in adding new ones. Blyth Software is the proper organization to contact with bug reports and suggestions for new features.

By the same token, Polymath creates the OmniScience newsletter. Although Blyth has a review committee for this publication and many Blyth employees have contributed their comments, I remain solely responsible for the content of the newsletter. (Blyth personnel may, from time to time, submit articles.) Responses to articles and suggestions regarding the newsletter should be sent to Polymath.

Technical support for OMNIS 7 can be purchased from either organization. Support from Blyth includes maintenance releases of OMNIS and may include discounts on future software versions. Support from Polymath cannot include software releases, but may provide answers to more difficult development and design problems using OMNIS. Such is the role of the consultant. Both avenues should be used, as appropriate, by the serious OMNIS programmer.

Welcome

continued from page 1

Volume I will continue to be published until the seventh issue, when all outstanding subscriptions will have been fulfilled. In response to concerns expressed by original subscribers who paid the price of subscribing to that limited edition publication, most of the information in those issues will not be repeated in Volume II. For those of you who missed out on the first volume, there are still back issues available (see the Products section later in this issue) and you may find them to be worth the investment. If enough interest is shown in a publication that focusses on design and consulting issues, a product named OmniBusiness will appear when Volume I is phased out.

The purpose of OmniScience is to further clarify and explore the details and nuances of issues involving the use of OMNIS. I have the luxury of being able to spend the time to thoroughly test all aspects of a feature and consider the implications of what I discover, and the space to write about these discoveries in detail for the benefit of all.

The purpose of this publication is certainly *not* to compete with nor to replace the manuals that are supplied by Blyth. They do an excellent, concise job of describing the features of OMNIS 7 and are the equal or superior of manuals for any other software package. If I occasionally give an alternate point of view from that expressed in the manuals, it is generally because a manual doesn't have the time or space to discuss nuances and style.

If I point out features that don't yet exist in OMNIS, I am not pointing fingers and denegrating OMNIS or

Blyth. If OMNIS weren't the best product available, I wouldn't be working with it. But if there are tasks that need to be performed, it is my charter to show you how to accomplish that task. Features evolve in OMNIS at a tremendous pace. For example, the *fld()* function was introduced in Omnis 3 Plus. Its counterpart, *Calculate with fld()*, didn't appear until Omnis 3.3. There are a number of command sets that are not yet "complete" in OMNIS 7, but the people at Blyth can only do so much.

There is a programmers' disease called "Creeping Elegance", a tendency to not release a project until it is "perfect". I am pleased that Blyth Software has not succumbed to this disease, because I wouldn't have OMNIS 7 to work with *now!*

Not all articles in OmniScience will apply to everyone, although I try to make them interesting to all. Some things that need discussion will only apply to Mac or to Windows users. Others may apply only to the US and Canada or only to Europe or Australia. If you don't use SQL, or a specific SQL platform, I must still write about these things for those who do. Please look over these other articles, they may still contain useful information for you.

I would like to thank the current management of Blyth Software for acknowledging the work I have done over the last seven years in educating OMNIS users around the world through their act of providing an OmniScience subscription to all registered OMNIS 7 users. This alliance between Blyth and Polymath can only be beneficial to all concerned. Blyth has created the best database development environment and Polymath provides in-depth information on how to use it. What could make a better team?

OMNIS 7 Class Schedule

For the past seven years, my primary commitment to the OMNIS user community has been in providing comprehensive and affordable education experiences across the United States. The articles and books I have written on OMNIS products, as well as this newsletter series, have all sprung from the needs expressed by the students I have helped. This tradition of assisting OMNIS users through traveling seminars continues.

A limited schedule of classes began earlier this year with a week-long series I taught in-house at Blyth Software in Foster City. Now that this newsletter is completed, I can offer a more aggressive schedule. Here are the courses I am currently offering.

Professional Developer Training

This three-day course is the prerequisite for all others I offer. It is a detailed look at the basic programming tools in OMNIS 7, including Connections, the CRB, Lists, Reversible Blocks, Event Management, and other issues. \$750.00 (auditors — \$250.00)

Advanced OMNIS 7

This two-day course delves deeper into the tools available in OMNIS 7. The Debugger, format and local variables, advanced List manipulation, and Ad Hoc reports are among the topics featured. \$550.00 (auditors — \$200.00)

Streamlining for Performance

This one-day course deals with issues of speed and efficiency in using OMNIS 7. Examples are taken from real world applications and students may pose their own in the afternoon session. \$300.00 (auditors — \$100.00)

Logistics

In each city to which I travel for these classes, I give one class of each course. Professional Developer Training is given Monday - Wednesday, Advanced OMNIS 7 is given Thursday - Friday, and Streamlining for Performance is given on Saturday. Each day is a full 8:30 am to 5:00 pm day. The classes are generally given in a hotel near a major airport for the convenience of students who fly in for the training. Classes are on a "bring your own computer, buy your own lunch" basis to save you money. You must also supply your own copy of OMNIS 7.

I offer a discount to people who take multiple full-paid classes in the same week and who pay for their classes two or more weeks in advance. This discount is currently \$100.00 per *additional* class. If I cancel, a full refund is given.

Students who feel the need to refresh their memory for a specific course they have already taken may audit that course at a reduced rate. That rate is roughly one third of the normal class fee. I have found over the years that auditors often learn more the second time around because of the repetition.

You may register by telephone, mail, or fax. Just send us the information requested on the Fax Survey on page 39, indicating which classes you wish to attend in which city from the schedule on the following page and include payment information. We hold all payments until we are certain that the class will be held. We need a minimum of eight full-paid people per class. (Auditors count one third.)

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 of OmniScience. I recognize that not everyone will feel the need for these, so I have made them optional.

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 four 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..."). These "Solution Packs" (as they will be called) will contain code and documentation on topics that cover the basic needs of business transaction database applications. For example, the first will focus on General Ledger accounting systems, the next on invoicing systems, and so on. Programming modules that you can patch into your applications *with no additional royalties* are included. A licensing agreement will accompany each Solution Pack basically stat-

ing 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:

Event Management Techniques

Advanced List Manipulation

Uses of the Debugger

eval() Explorations

DMS to Decimal Conversion

and many other related techniques.

This disk is priced at \$20.00 — order yours soon!

Also available this issue

OMNIS 7 Class Schedule

For more information, call 510-522-6107.

	Prof Dev	Advanced	Streamlining
Los Angeles	Apr 6-8	Apr 9-10	Apr 11
Boston	Apr 27-29 (follows MCN Conference)	Apr 30-May 1	May 2
Chicago	May 11-13	May 14-15	May 16
Oakland, CA	May 18-20	May 21-22	May 23
Australia	June (dates and locations TBA) (major city tour following OMNIS Developer conference)		
Oakland, CA	Jul 13-15	Jul 16-17	Jul 18
Detroit	Jul 20-22	Jul 23-24	Jul 25
Boston	Aug 10-12 (follows MacWorld Expo)	Aug 13-14	Aug 15
Seattle	Sep 14-16	Sep 17-18	Sep 19
Oakland, CA	Sep 21-23	Sep 24-25	Sep 26

OMNIS 7 Shell Application
\$150.00

A great starting point for any application. It will save you hours on each new project! Open code.

Black Boxes
\$30.00

Modular procedures for data conversion that you can paste into any application. Open code.

Data Generator
\$50.00

If you ever need massive numbers of records for testing your applications, this application can give them to you. It creates real-looking random client records. (Don't mail to them!)

Unlocking Omnis 3 Plus
\$49.95 (plus shipping)

Many people are still actively using this version of OMNIS. This is the only viable reference available if you are.

1st Quarter Fax Survey

I can give you better information if I receive some from you. This survey is intended to find out what kind of information OmniScience readers want as well as what kind of OMNIS support products and services you feel are needed. I would also like your feedback on the articles presented in this issue.

Please mail or fax your completed survey to:

OmniScience
 1418 Park Avenue
 Alameda, CA 94501
 Voice: 510-522-6107
 Fax: 510-522-6110

Please check the class site(s) that would be most convenient for you. If another major city is more convenient, please write it in the blank below.

I have no need for training.

United States _____

- Anchorage
- Honolulu
- Seattle
- Portland
- San Francisco
- Los Angeles
- San Diego
- Lk Tahoe/Reno
- Las Vegas
- Salt Lake City
- Phoenix
- Albuquerque
- Denver
- Dallas
- St. Louis
- Minneapolis
- Chicago
- Memphis
- New Orleans
- Detroit
- Columbus
- Philadelphia
- Buffalo
- Boston
- New York
- Washington
- Charlotte
- Atlanta
- Orlando
- Miami

Australasia

- Brisbane
- Sydney
- Melbourne
- Adelaide
- Perth
- Hobart
- Auckland

Europe

- London
- Paris
- Lisbon
- Madrid
- Berlin
- Oslo
- Geneva
- Amsterdam

Canada

- Quebec
- Toronto
- Calgary
- Vancouver
- _____

Latin America

- Mexico City
- Cancun
- Quito
- Buenos Aires

There are dozens of classes that could be given on OMNIS 7. Here is a list of the most common requests. Please let me know which of these would interest you in the future.

- Introductory (Fundamentals)
- In-depth basics (Prof Dev)
- Advanced
- Database Design Techniques
- Interface Design Techniques
- Report Techniques
- Externals Workshop
- OMNIS and SQL Platform _____
- OMNIS and IAC
- OMNIS and DDE

The quest for information didn't leave much room for orders. Please indicate any of my products you would like to order or classes you wish to register for here. We will call you to complete the order.

Name _____

Company _____

Address _____

City _____ State ____ Zip _____ Country _____

Voice phone _____ Fax _____

Please rate the major articles from this issue of OmniScience.

	Useful	Clear	Too simple	Too advanced
Events	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Lists	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Debugger	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
eval()	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
jst()	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ad Hoc	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
T&T	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Comments _____

Below is a list of OMNIS support projects that I have in the planning stages. Please indicate which of these would interest you.

- An OMNIS 7 book
- Design mag (OmniBusiness)
- Audio instruction materials
- Video instruction materials
- Finished program modules

Please (circle one) send / fax me more information about

- Classes available
 - OMNIS support products
- The disk type I require is
- Mac PC 3 1/2 PC 5 1/4

Next Generation

continued from page 1

Often, manufacturers of database programming languages try to convince the public to buy their products by basically *lying* about how “easy” it is to create complex applications with their software. A major point in Blyth’s favor (in my book) is that they have never taken this approach. The new management at Blyth makes no bones about the fact that database application development can be a complex task and they are open to working with experienced professionals to help solve problems for their clients.

Yet, in spite of the complexities involved, OMNIS 7 makes this task more of a joy than any other development environment I have ever used. I have watched OMNIS evolve over the last seven years and have seen some of its earlier incarnations created in the late ’70’s and early ’80’s. The technological leaps have been impressive! Omnis 3 was the first viable multiple file database for the Macintosh. Omnis 3 Plus was a quantum leap beyond 3.

Omnis Quartz was the first database product on the Windows platform. Omnis 3.3 and Quartz 2.0 could share data on a mixed multi-user network of Macs and PCs and were the first on either platform to speak SQL. Omnis 5 allowed easy transportability of applications from Macs to Windows with a minimum of conversion required and enhanced SQL access.

Now OMNIS 7 has arrived. It was *completely rewritten* in C++ — an impressive feat in itself! Applications can now be *shared* by Windows machines and Macs *with no conversion necessary* — and UNIX is right around the corner!

No product is perfect, and OMNIS has room to grow like any other. But, over the years, whenever a need has been expressed for a new feature, the programmers at Blyth have come through with much more than just the requested feature. In this newsletter I may occasionally mention a feature it would be nice to have and a way to use current features to perform that task. I have no doubt that future enhancements to OMNIS will include many of these proposed additions.



OmniScience is published quarterly by Polymath Business Systems, 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): 4 issues, \$75.00. Canadian subscriptions, \$80.00. Outside the USA and Canada, \$90.00. Single issues are available for \$20.00 domestic, \$22.00 Canada, and \$25.00 elsewhere. Send subscriptions, changes of address, and fulfillment questions to *OmniScience*, 1418 Park Avenue, Alameda, CA 94501.

Copyright © 1992, 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.

OmniScience and Polymath are trademarks of David Swain. OMNIS and OMNIS 7 are trademarks of Blyth Software, Inc. Microsoft and Windows are registered trademarks of Microsoft Corporation. Other software and hardware mentioned are trademarked by their respective manufacturers or distributors.

In the Next Issue

A lot of the traditional methods of working with OMNIS have changed with the introduction of OMNIS 7. Volume II Number 2 will begin the analysis of some of these changes and offer some well-tested suggestions. And then there’s SQL...

Prepare for Update

Once a delicate flower that might wither at the slightest hint of programming impropriety, PFU is now a robust mode that won’t quit unless you make it. Now there are new ways of getting into trouble.

Format and Local Variables

Subroutines just aren’t what they used to be. Format and local variables open up new possibilities for structuring your applications. We’ll explore a few.

lookup() Function

This is vying for a place among my favorite OMNIS functions. Access to other datafiles is just the beginning!

Storing Lists

Lists aren’t just temporary repositories for collections of values anymore. We’ll explore storage, retrieval, and reporting techniques for Lists stored in the datafile.

OMNIS 7 and Sybase

Sybase has some special features not shared with other SQL platforms that make it a great choice if you need a client/server database. Here are the ways OMNIS 7 takes advantage of what Sybase offers.

SQL Reports

There is no direct reporting capability for SQL databases built into OMNIS 7, but there are a number of ways to generate reports on data retrieved from the server through OMNIS.

Tips and Techniques

More quick tidbits to make your programming life a little easier — or, at least, more interesting!