The International Journal of OMNIS™ Developers

A Publication of The DLA Group Pty Ltd

Omni Directional

7

July/August 1997

DLA Launches OMNIS Studio in Australia

SYDNEY - Tuesday 12 August, **David Lewis**, CEO of DLA; **Pat McEntee**, VP of Marketing, OMNIS Software, Inc and **Kinley Pon**, Senior OMNIS Evangelist, OMNIS Software, Inc conducted the Australian launch of the world's first crossware development tool, OMNIS Studio.

A packed audience heard from Pat McEntee how OMNIS Software has been re-born both technically and financially and were inspired by the technical wizardry of Kinley Pon. The verdict of all attendees was one of great optimism.

OMNIS Studio is the first Rapid Application Database Development tool to deliver crossware, software that is cross-platform, cross-database, cross-object and cross-architecture. OMNIS claims that no other tool provides the power to do so much with all component types.

OMNIS Studio is a complete component design and Web integration system that enables the importation, assimilation and extension of objects including Java, ActiveX, OCX, VXB, C++, OMNIS and other native component types. Exciting new Web tools are now available within OMNIS.

DLA has generated significant interest in OMNIS Studio since the launch and several new leads have been generated. Support for these tools is evident from outside the current OMNIS developer community and there is now real scope for an expansion in developer numbers.

In a rare show of emotion, the Australian OMNIS developer community gave Kinley a standing ovation at the end of his Studio presentation. Of greatest interest was the demonstration of Java and ActiveX combining in the one application. OMNIS claims to have ended the 'war' between Sun and Microsoft by delivering this solution.

Renewed Sales

As a result of the launch, DLA's OMNIS sales team has been swamped with enquiries and new leads. A significant number of developers embraced the special Australian discount, an offer not available to any other developers in the world. Whilst developers were understandably cautious of the new technology, all agreed that OMNIS Studio was the most significant software development from OMNIS in years.

Stephen Farmer, long time OMNIS developer and Managing Director of HTS Pty Limited said: 'OMNIS Studio is an essential requirement for my team of OMNIS developers. We have ordered both MacOS and Windows platforms and we will commence development immediately'.

Jed Wesley-Smith, OMNIS evangelist with Sydney based Objectif Pty Limited was also enthusiastic. 'I am very excited by OMNIS Studio. This development environment will enable us to compete with the best and deliver state of the art applications to our clients. At last the best RAD (rapid application development) environment is available in a true object oriented form.'

Australian OMNIS World

DLA has planned many new advertising initiatives and expects to host our OMNIS World conference in Sydney in early December. A large US based OMNIS delegation will attend and several streams are under preparation. This conference will also target the growing South East Asian region.

Far Left: *Kinley Pon, OMNIS Software (centre), Dr John Lambert, of Deltra and James Coroneous, HAS Solutions at the OMNIS Studio Launch.*

In This Issue...

OMNIS 7 version 3.6

This release note describes the New features and Fixes in OMNIS 7 version 3.6, any Notes and Cautions for this version and interim releases, and the validated DAM configuration for each platform. The Fixes and Notes contain a reference number such as "GR/RU/412FC" which identifies the problem in the OMNIS Software faults database.

New Features and Enhancements

Informix support for SShell, DShell, and VCS

OMNIS 7 version 3.6 contains two versions of the DShell and SShell: the first version, which is installed by default, supports Sybase and Oracle only and can be used to access new and existing Sybase and Oracle VCS repositories; the second version supports Informix as well. If you want to use the new Informix-aware shells you must install them using the Customize option in the OMNIS installer, replacing the default DShell and SShell. Note that the new Informix-aware shells support new VCS repositories and connections on Sybase and Oracle as well as Informix, but you must not use them to access existing Sybase and Oracle repositories created using earlier versions of OMNIS.

OLE2 support

This version installs the OLE2 files by default.

New FileOps external commands

Four new commands are added to FileOps: Open resource fork, Truncate file, Get files, Get folders. The latter two were contained in the DIROPS external package which has been removed in this version.

New sys() functions

sys(8) returns the platform type of the current OMNIS program as a string: 'MAC68K', 'MAC600', 'WIN16', 'WIN32', 'OS2'

sys(120) returns true if small fonts are in use.
sys(121) returns true if large fonts are in use.

Notes and Cautions

GR/@B/064FC Platform: Windows NT only
If your connect protocol is named pipes or IPX/SPX you may have problems retrieving multi-column data that includes a binary picture field. Data corruption can occur when all characters of the server column are filled, that is, the last character of the text is replaced with a garbage character. For example, "Dumbo" would come back as "Dumb|", the last character being "|", the vertical bar character. The Picture field is retrieved intact and displays correctly when viewed. If the binary picture field is not included in the data retrieval, this type of corruption does not occur.
This problem only occurs if the connect protocol is named pipes or IPX/SPX. These are not supported or tested protocols. TCP/IP will not cause this problem, so ask your Database Administrator to modify the MS-SQL 6.0 server to allow connections via TCP/IP. To do this:

–Open the SQL-Server 6.0 tools on the client machine.

–Open the Client Configuration Utility and choose Advanced.

–Select the server name and change the DLL to TCP/IP. Add the IP address and port number of the server in the connect string field.

–Click add modify, and click on Done.

GR/DC/191FC Platforms: All
Under the Help system topic "Combo boxes" it incorrectly states that you can expand a combo box by pressing the spacebar. However, this is true for dropdown menus only.

Fixes, Warnings and Notes

A more detailed version of this article, together with *all* the fixes introduced with this version of OMNIS, complete with cautions about its use, validated DAM configurations and other important information may be downloaded from: ftp://ftp.dlagroup.com.au/pub/Omnis/Documentation/OMNIS_7v36_readme.txt.

Omni *Directional is produced by DLA Technology as a service to users of Omnis Software's Omnis™*

D L A
TECHNOLOGY MANAGEMENT CONSULTANTS

A Bi-monthly publication of:
The DLA Group Pty Limited
A.C.N. 003 329 039
Level 3, 35 Clarence Street
Sydney NSW 2000 Australia
Tel: (+61 2) 9262 2255
Fax: (+61 2) 9262 2290

DLA Technology
- Change Management
- Database Design
- Consulting Services
- Software Development
- Internet Advertising
- OMNIS Software distribution and development

DLA Software
- Solicitor's Companion practice management software for solicitors
- Counsel's Companion practice management software for barristers
- Litigation support development and services
- Databases for marketing, human resources and library management

Email:	info@dlagroup.com.au
URL:	http://www.dlagroup.com.au
Editor:	Daniel Lewkovitz daniel_lewkovitz@dlagroup.com.au
Managing Director/ CEO:	David P. Lewis david_lewis@dlagroup.com.au
Submissions:	omnidirectional@dlagroup.com.au odsubmit@dlagroup.com.au for submission guidelines (autoresponder)

Omnis and Omnis 7 are Trademarks of Omnis Software Inc. All other product or service names mentioned herein are trademarks of their respective owners.

Copyright (c) The DLA Group Pty Ltd 1997. All rights reserved.

Old Faithful

Although no longer the focus of much debate in developers' circles, like a faithful old Victa lawnmower or Hills hoist, OMNIS 7v1 is still going about its tasks delivering worthwhile solutions to customers' problems. Stumble across a copy of OMNIS 7v1 in a garage somewhere, give it a pull on the rope and it will splutter into life. And while it may not be as flash as the latest Flymo Gizmo or Agitator De-Humidifier, when the job is over, the lawn is still cut; or the clothes are dry. And after all isn't that what the customers want?

In the year, 2007, Elvis will be rumoured to have been dead for thirty years but what will they be saying about OMNIS 7v1? If the stories about OMNIS 3 folk offered in the course of preparing this article are any guide, it may well be "OMNIS 7v1 is alive and well and still impressing people".

Where will the computing industry be and in what direction will it be headed? Using more resources to do pretty much the same task if recent experience is any guide. Apple recommends 12 Meg of RAM be available for OS8 (at least their figures bear a more passing resemblance to reality than Microsoft's published estimate of 4 Meg for Windows 95). And most of us have been around long enough to remember 128K Macs, if not without total affection. But are we nearly 1,000 times more impressed? Do we no longer value doing more with less?

How much comparatively legacy equipment is still in daily use? While the price of computing resources is falling, they still aren't free and there is the issue of obsolescence - making the case for trashing something that still works while there are other claims on the resources required to. If the job can be done adequately with fewer resources, doesn't it make sense for the solutions to be more widely distributed rather than add some fancy bits and keep deployment of the solution to the inner sanctum of the comparative privileged? Despite its power and considerable version modification, on their own figures, more than 90% of Lotus Notes users use it for email. OS/2 might be technically excellent but…? And is Windows 3.1 still in use anywhere by any chance?

And why is just about everything in the database field more resource hungry than OMNIS 7v1? These days, which product lets you rock up to a client site, with a developer copy of your database engine, your app and a demonstration datafile on a single disk; and then installs without creating a lot of tedious files in strange places? What is the cross over point between a developer which urges the implementation of the latest Gee Wizzery and the bill playing client who barely understands the difference between a megabyte and a mozziebite? If one moves in developer circles too exclusively, it is easy to overlook these bread and butter issues.

Other database vendors seem to have entered the GUI market with higher end screen controls in place but coming up through the ranks of OMNIS 5 and 7, Omnis software offers a tested, robust solution whose competitive advantage is its economy. Oracle quotes statistics which claim 70% of U.S. homes and 90% of homes worldwide don't own a computer. While databases are not typically home use applications, these figures are an indication that when it comes to the big picture, computer penetration may not be as pervasive as those who move in those circles may tend to think. This could well be the market for the next category of applications.

OMNIS 7v1 circa 1998!

So what are people doing with OMNIS 7v1 these days? If you thought "Petl", "Speleotechnics", "Grivel", "DOBI" and "Grips" were the names of some more imaginatively chosen temporary variables, they are in fact the names of leading climbing and caving equipment manufacturers. OMNIS 7v1 applications drive the activities of Spelean, their Australian distributors helping to make sure that which goes up (or down) comes down (or up) again safely. When he's not wearing a hat with a light on the top, Managing Director Andrew Pavey dons one with a propeller to conduct meetings at Sydney's Club Mac. Andrew has a couple of Power PCs for publicity and presentation type work but, when it comes to real work, he's still using some older 680x0 Macs and not having to upgrade them pleases those who find descending to the bottom line far enough for them.

Some respondents to my OMNIS 7v1 research listed large numbers of applications they have been able to develop and most importantly support pretty much single handed. These applications covered uses as disparate as Conference Management, Student Housing Service Database, Distributor Database, Rostering System written for a large Telecommunications carrier, Safe Behaviour Tracking and so on. When being responsible for such a varied group of applications, customers really appreciate an application which does not require constant periodic care and maintenance. In many cases, applications of this type started in earlier versions of OMNIS and are still delivering the goods. As networks improve, hardware becomes more powerful and the upper end of the project is scaled up, it is easy to forget there can be a whole new group of applications coming in at the low end; applications which might not happen if high end approaches were applied.

And when one strays from the fold (probably because the client has read some half-baked articles in the computer press or fallen under the spell of a reputed expert) what can happen? A correspondent tells me of a sophisticated accounting application developed over some 30 months in 7v1 and in the end being used by up to 40 users on a 350 Meg native datafile.

With a couple of small things still needing attention, everything stopped and the components split into stand alone applications. And off the shelf program for renting, Peachtree for general accounting. More than 19 months later, the OMNIS app is limping but still working, no maintenance having been done during that time and on the other side a yawning black hole - no progress made. Netscape was to be involved as well but upgrading to 17" monitors (40 users remember).

When it comes to the database engine, Illustra was going to provide "excellent performance" but with a fully loaded datafile and 40 users doing their thing, OMNIS 7v1 gave comparable performance. Then there are issues of staff frustration and client aggravation. The motivation to move from something that worked was to finish up with something "better". But at what cost? And if it doesn't work, what are the chances of recovery in those relationships?

Satisfaction Guaranteed

It's now gone into folk law that several years ago an OMNIS customer was so frustrated with her experiences, she sought assistance on CompuServe to commence a class action. Well, it *was* OMNIS 7, you *are* entitled to a happy ending and there is one. A purchaser who taught herself Omnis 5 had got out of her depth, subsequently employed two separate developers and finished up with a corrupted datafile at version 7v1.2 by the time writs were called for.

When the alarm was raised, one of our trusty Listers, Alan Reinhart, <res@netreach.net> jumped on his Harley, flicked on the red and blue lights and on arrival found a datafile in what others had diagnosed to be a terminal state lying prone for about twelve months in the hallway where people had been stepping over it. Sensitive CPR was applied, another datafile segment added a re-index here and a re-organisation there and colour returned to its cheeks. The customer continues in her role as (frustrated) project manager often demanding things be done the hard way but OMNIS keeps delivering the goods but here is a rare customer who has escaped the clutches of the legal profession. Is that the *real* triumph of the story?

When it comes to pushing the envelope in an "old" version of OMNIS , the story from Nick Andritsakis <infomega@c-id.gr> probably can't be beaten. Nick became interested in applications dealing with motor racing results when he benchmarked his first AppleSoft program to generate a classification on a day's race results written for an Apple IIe against a mainframe operating over leased lines. The score: mainframe 1 minute, Nick 45 - there was room for improvement.

Richard Ure wallowing...

The Automobile Club of Greece learnt of Nick's interest and suggested they move from their mainframe based system to his emerging application. The number of accredited rallies was being reduced and organisers had to try harder to stay in the game. It was a big risk for Nick: hundreds of journalists distributed over two cities several hundred kilometres apart would be awaiting his output and when journalists have nothing to write about they aren't reluctant to take a vitriolic view of those responsible for informing them. On top of that, OMNIS 7 had just come out and Graphit looked promising - did Nick come from a small family of folk who jumped out of aircraft and then starting wondering about the location of the parachute?

1992 was his first year and OMNIS 7 was at v1.03 considered at the time to be "more stable". After many very long days, Nick thought he had everything covered. Unfortunately the immaturity of that version and normally reliable Apple hardware failures proved disastrous; leased datalines were too noisy; saved lists were used to help achieve the required speed and this produced some dreaded "end of file messages" in multi-user mode; time pressures limited the use of backups. Completely accurate results were produced and LaserWriter and Graph-it output assuaged some pretty unhappy and disappointed people but face had been lost. It was time to skin the cat from the other end.

The multi-user part of the application was completely re-written and this time there were three datafiles. The first was a data entry and results datafile with full multi-user access on Ethernet with all information needed by the Clerk-of-the-Course. The second, for the local press, received data in the form of ready made parsed reports kept in text fields while a summary field was used as a directory to the available reports. This update routine also dropped the parsed reports to text files in a fixed name directory.

The application running on the remote server concurrently with AppleShare would automatically scan this directory via a modem link at set intervals. By using FileOps extension, it would determine whether the third remote data file needed an update. Nick tells the story best in his own words:

"All terminals in the remote site would be hooked on this data file residing on the remote server and use the same list to choose one of the available (recently updated) reports. Switching and updating data files and exporting text fields on the server via Ethernet were done automatically in less than 10 seconds in our data entry LCs. It took a further 20 seconds for the remote server to import data via modem link and update the remote data file.

The overall performance was impressive.
10 seconds after the third car (for example) was finishing a special stage we had the available 3 times over the VHF communication.
10 seconds to enter them in Rally Master.
10 seconds to print a classification to screen and check it.
30 seconds to update everything.
So, within 1 minute after the third car (for example) was finishing a special stage 250 km away, every reporter could have a new classification on his reports list to print or see it "live" in a TV set"
The very next year Nick had much improved speed, rock solid software from Omnis-Software, Apple hardware back to its usual reliability.
Very close to client server in 7v1 where it still is, four years later.

Conclusion

OMNIS 7v1 is still a most viable option for the committed OMNIS developer with space and equipment restrictions. Many clients outside of the US do not immediately embrace the latest software for at least 12 months after its release and this cautious approach is certainly repeated at the corporate level as well.
The DLA Group still supports OMNIS 7v1 and the development version and runtimes are still available.
Richard M. Ure
richard_ure@dlagroup.com.au

Shadow Files

There are times when it is convenient — even necessary — to store information derived or condensed from the basic data of an application in separate data structures in the same datafile. For example, we might find benefit in storing *summaries* of transactions in a sales tracking system as well as the detailed transactions themselves. At first glance this appears to be a direct violation of traditional database design rules (storing data redundantly), but it can be a very powerful technique if used properly and under intelligent control.

I call the structures used to track such "redundant" data "Shadow Files" because they "follow" the basic data of the application like a shadow. Their records are created and updated in the background, quietly, as other processes are taking place. They require only microseconds of time to be maintained, yet they can be of enormous value when they are needed. What statistics lurk in the hearts of your applications? The Shadow knows … if you have programmed your applications to keep the Shadow current!

The Purpose Dimension

Let's clear up this "violation of rules" misconception before we go any further. Years ago I added a small section to the OMNIS database design classes I teach where I introduce the concept of "purpose" as a dimension, or attribute, of data. Identical field values with different purposes are, in fact, different data and *not* redundant, even though they refer to some aspect of the same entity. My examples in class have to do with redundant use of fields in different Files, but storage of derived data would also come under the umbrella of this explanation.

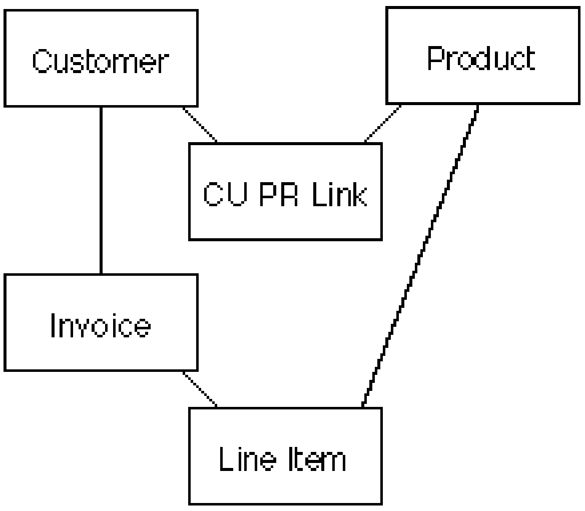
The classic example is the need to redundantly store a customer's address in the header of an Invoice (even though it is already stored in the Customer record) because that address could change over time. Real world considerations (sales tax audits, etc.) require that this information be fixed or frozen into the Invoice record for all time as insurance against a possible move on the part of the customer. The purpose of the address in the Invoice File (the address to which products were sent for that transaction) is separate from that in the Customer File (the customer's current address).

In a similar way we can use background processes to update summaries of transactions that are important to our application even though it might seem on the surface that these summaries could be derived through some reporting process. Occasionally such summaries become more important over time than the detailed data from which they can be derived — and the derivation becomes more tedious and time-consuming in the process. In fact, it may be necessary or desirable to archive and purge old detailed data, but retain the summaries! As always, circumstances, client needs and performance issues override strict adherence to impractical academic "rules" of database design.

Following this logic, Shadow Files are not really repositories of "redundant" data, but are constantly updated summaries or composite views of certain aspects of the data that may be difficult (or even impossible) to recover by other means, but that are very important for the owners of the data. Their purpose may be a transitional step in a more involved process (summarizing sales transactions for a specific period — a week, say — to allow for a single significant, and automated, journal entry rather than thousands of little ones) or an end in itself (archiving of monthly activity statistics). The actual reasons for employing Shadow Files will vary widely depending upon circumstances, but will always come down to real world needs taking precedence over pedantic and simplistic rules. I am *not* recommending that you exercise the "just because I feel like it" dimension in your work (although who am I to stop you?). Certainly one can justify *anything* with enough convoluted logic. The use of Shadow Files is a discipline like all other aspects of database design. Hopefully the examples I give in this article can help you form a set of guidelines for your own use of this technique. At the very least, I hope to open you to some of the possibilities.

Purchase History example

Consider this example from the sales field. (If you don't write sales systems, see if there isn't a similar problem in your chosen field.) Suppose your company sells consumable goods that customers order by telephone (or, perhaps, they are contacted periodically by your telemarketing staff). There are many products available and it is hoped that most customers will become regular repeat customers for at least a few of them. Part of the job of the telemarketing staff is to encourage continued use of products already used and to suggest similar or complementary products for customers to try. It would be very convenient for the telephone salesperson to be able to see what products a specific customer has ever purchased, when each product



was first purchased, when each was last purchased, and what kind of quantities were involved to assist in their sales effort with that customer.

Generating such a product-by-product summary from past invoices, as either a screen report or a list on a window, could become a lengthy process as your datafile grows — especially for your best customers! You can imagine the problems: Gathering all line items from all past invoices for that customer, sorting that collection by product and purchase date, and then consolidating that information and presenting the summary on-screen.

It is much more efficient to gather this information as it is being stored in the original source documents. In addition to our normal four File system for the sales transaction (Customer, Invoice, Line Item, and Product Files), we could add a Customer-Product Link or Purchase History File connected to Customer and Product to store this information. We simply need to determine what fields are required and at what point we want records to be generated and/or updated in this new File.

In the simplest of cases we would need six fields to track this information. The first is a key field that combines the Customer ID Number and Product Code into a single indexed value. This will allow us to very quickly locate a specific customer-product record or determine that one needs to be created. Next we need a field for the original purchase date and the most recent purchase date (both indexed). We should include fields for the cumulative number of invoices on which this product has appeared for this customer and the cumulative number of units they have purchased. From this information we can derive the average number of units per order. Finally, we need a field for the number of units purchased on their most recent order. With reasonably-sized customer and product codes, and using short date and long integer fields, a customer-product link record will only require about 60 bytes including index entries.

Setting up the structure to hold this data is no problem — determining the rules for *managing* this data is the real trick. We have to think in real-world terms … and the real world is not simple. We *can*, however, impose a certain rigidity on our business practices to make the task simpler. If we allow *any* invoice in the system to be modified at *any* time, we will find ourselves run ragged trying to track down all the possible ways a user might need us to back out and otherwise update our summary information. But the real world has identifiable limits that we can use to tighten our control of this situation.

Most real businesses recognize a point in the history of a transaction where that transaction is considered "complete" and unalterable. This may be the point at which the order has been entered and an order number has been generated, the point at which the products have been shipped (either the initial shipment or the shipment that completely fulfills the order), the point at which payment is received, etc. Whatever that point, changes are no longer allowed to the invoice. (Exceptions such as returns or late changes are dealt with using Return or Change Orders after that point.) *This* is the point at which we will update our Customer-Product Link File with the information from our invoice. (If your application needs to track the number and nature of changes to a record, that is a different problem — but still a use of Shadow Files!)

Our Invoice Header File needs a field to hold its status, as discussed above, so that our code can know whether changes are allowed and when to update the appropriate Product History records. I usually use a *Short integer* field for this purpose, since there may be a number of status levels (but fewer than 257). I manage the values this field takes on with background processes rather than with radio buttons because I don't want my operators to "accidentally" revert an invoice record to a previous state (a privilege reserved for the system administrator). I also offer shortcuts, such as a No/Yes message that pops up on clicking OK from the insert and edit window control procedures asking if the operator wants to "commit" the invoice, to streamline the process. There will also be a "Commit" button on the toolbar to deal with a "pending" invoice that requires no changes but is now ready for delivery.

Whatever method is used to complete the invoice, a section of code will traverse the line items and, for each line item, will call the following subroutine chain. Bear in mind that the appropriate Customer and Product records are already in the CRB when the call is made.

```
Product usage
——————————
Begin reversible block
        Set main file {CU PR link}
        Set read/write files {CU PR link}
End reversible block
Single file find on CPL_CODE (Exact match)
{jst(CU_ID_NUMBER,9,PR_CODE,5)}
If flag false
        Calculate CPL_CODE as
jst(CU_ID_NUMBER,9,PR_CODE,5)
        Calculate CPL_ORIG_DATE as IV_DATE
        Call procedure Usage update calcs
        Prepare for insert with current values
        Calculate IT_CPL_POINTER as CPL_CODE
        Update files
Else
        Prepare for edit
        Call procedure Usage update calcs
        Calculate IT_CPL_POINTER as CPL_CODE
        Update files
End If

Usage update calcs
————————————
Calculate CPL_LAST_DATE as IV_DATE
Calculate CPL_LAST_QTY as
IT_QUANTITY*pick(IT_CASE_OR_UNIT,PR_CASE_QTY,1)
Calculate CPL_TOTAL_ORDERS as
CPL_TOTAL_ORDERS+1
Calculate CPL_TOTAL_QTY as
CPL_TOTAL_QTY+CPL_LAST_QTY
```

Notice that I use a *Single file find* rather than a *Find* to test for the existence of the Customer-Product Link record. If I had used a *Find* command, both the Customer and the Product record would disappear from the CRB if the test fails. This way the proper records in both Files are available should the link record have to be created. The library I borrowed this bit of code from was written to allow either "units" or "cases" to be the quantity unit for line items. For the purchase history, though, I felt that it was simplest to convert this quantity to units. The second line in the "Usage update calcs" subroutine performs this nicely.

Also note that, in the routine that called the "Product usage" subroutine, the Line Item File was already set to read/write mode, so its fields can be updated in these subroutines without *r*esetting that mode. If our telemarketers will also need access to the detailed line items from which a customer-product summary is derived, we must "backfill" each Line Item record with the customer-product code so they can quickly be built into a list. This field in the Line Item File must be indexed.

If we need to monitor product *returns* as well as purchases, appropriate fields for return dates and quantities must be included in our customer-product field list (as well as a structure for tracking multiple line item return orders). Again, depending on real-world needs, this system can grow in complexity. Our job is to keep it lean and efficient.

Using the information

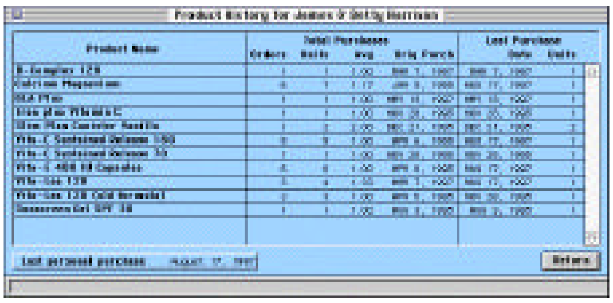
The whole purpose of the preceding discussion was to create code that gathers information about past product purchases that can be quickly and efficiently displayed for the user as telesales conversations are being held with customers. Now we need to examine how this information display takes place.

We simply build a list of all the customer-product records connected to the current Customer record and open a window to properly display the results (unless no sales have been completed to that customer). This process would be initiated from our basic Customer Information window and will only be allowed if a Customer record is in the CRB.

The display window for this example will open on top of the Customer Information window, so the code is written in such a way that the Customer record remains in the CRB and that the Customer File will remain the Main File once the procedure is completed. This way the program will act as though the Purchase History window had never been there once it is closed again.
Client needs will dictate how the purchase history list should be sorted. For this client the list was categorized by Product Type (an integer field) and then by Product Name. (The "Type" is not actually displayed, but operators of this application are familiar with the five basic product types of that company and know to which type specific products belong.)

Here is the code:

```
Purchase history
————————————
If not(CU_SEQNO)
        OK message {There is no Customer record on
screen for whom to view a Purchase history.}
        Quit procedure
End If
Begin reversible block
        Set main file {CU PR link}
        Set current list #L4
        Clear sort fields
        Set sort field PR_TYPE
        Set sort field PR_NAME(Upper case)
End reversible block
Define list {CU PR link,PR_CODE..PR_NAME,PR_TYPE}
Build list from file on CU_SEQNO (Exact match,Use sort)
If not(#LN)
```


Product History for James & Betty Harrison

Product Name	Total Purchases				Last Purchase	
	Orders	Units	Avg	Orig Purch	Date	Units
B-Complex 125	1	1	1.00	[illegible]	[illegible]	1
Calcium Magnesium	6	7	1.17	[illegible]	[illegible]	1
[illegible]	1	1	1.00	[illegible]	[illegible]	1
[illegible] Vitamin C	1	1	1.00	[illegible]	[illegible]	1
[illegible]	1	1	1.00	[illegible]	[illegible]	1
Vit. C [illegible]	5	5	1.00	[illegible]	[illegible]	1
Vit. C [illegible]	1	1	1.00	[illegible]	[illegible]	1
Vit. C 400 [illegible] Capsules	6	6	1.00	[illegible]	[illegible]	1
Vit-Gen 120	5	5	1.00	[illegible]	[illegible]	1
Vit-Gen 120 [illegible]	2	2	1.00	[illegible]	[illegible]	1
[illegible]	1	1	1.00	[illegible]	[illegible]	1

Last personal purchase [illegible]

Return

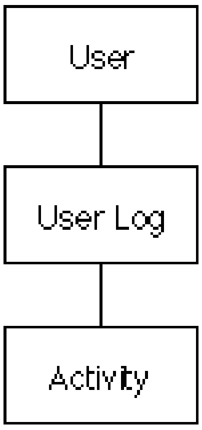
```
        OK message {No sales have been recorded for
[CU_NAME]}
        Quit procedure
End If
```

```
Open window Purch history/CENTER
```

The resulting list is displayed in the window shown in Figure 2. The field used for the "Avg" column is simply a no-name display field calculated to divide total units by total orders and formatted to show the result as numeric data with two fixed decimal places. A table field was used here to represent the list because of the formatting flexibility it provides. The vertical lines that separate groups of columns are created by holding down the Shift key and then clicking and dragging the left edge of the table field. If you need the line to extend through the column heading section, then it must be created from the edge of that section. If it is created from the edge of the record section of the table field, it will only divide columns in the body of the list.

Should you need to display more information than the width of your window can accommodate, you can allow scrolling under leftmost column divider. You simply add a horizontal scroll bar to the table field and turn on the *Fixed left column* attribute of the field. a vertical column divider must be used for this attribute to take effect, but you can make the divider invisible (if you so desire) by selecting it and choosing the line style labeled "none".

To wrap up this example, the Purchase History records are separate from, but dependent upon, the actual Purchase Transaction (Invoice and Line Item) records. In this case, the Purchase History information could be derived directly from the source data, but not quickly or efficiently enough for our intended use. The small amount of extra storage (and programming) involved pays tremendous dividends in increased productivity for our end users. Now let's examine a slightly different case.



Activity Log example

In many large database installations, there are many uses for tracking record modification information – that is, what records did a specific user create, modify or delete in a specific logon session. This may serve as a cross-check of productivity in a telemarketing application or as a security or administrative measure in other situations. (It's amazing how popular this item is with companies facing large layoffs, pending strikes, or other major business transitions!) This is *not* information that would normally be stored in the datafile as it is not the focus of the application. Nor can it be derived from any existing data. It *is*, however, valid data from the real world that can serve a practical purpose if properly captured and retrieved. I use this as an example here because it is another application of the Shadow File concept.

Basic Design

This technique relies on the existence of a user-based password protection system for the application. There is not time here to go into all the details of such a system – suffice it to say that there exists a User File that has a uniquely indexed User ID field. In most of my applications of this technique, I also include a User Log File (connected to the User File) that tracks the sessions of each specific user.

Each user must pass a logon window to gain further access to the application. The RSN values for both the User record and the User Log record generated by the logon are stored in #60 and #59 respectively as part of the logon process and are available at any time during the session for retrieval of those records. Access to certain parts of the application or to certain commands may be restricted for a user depending on their type and authorization level (as maintained in their User record by the system administrator).
Connected to the User Log File is the Activity File. This File contains three data fields: a date-time field, a "type" field (where 0="insert", 1="edit", and 2="delete"), and a field that is a composite of the File Format name or number (the latter taken from a Data Dictionary File, but only if you have one) and the RSN of the record effected. The first and third fields are indexed.
Implementation

Whenever a record is created, modified, or deleted, a subroutine in the STARTUP menu (or some equally convenient place) is called as part of the process. (In the case of multi-line transactions, usually only changes to the transaction header File are tracked.) The subroutine is sent three parameters: the name of the Main File, the Main File RSN, and the "type" of datafile change just made (or just confirmed and *about* to be made in the case of a deletion) to that record. The Main File and sequence field names can either be hard coded for each instance or derived using the *sys(82)* function and appropriate notation. The "type" value must be hard coded in all cases.

The activity tracking subroutine then creates a new record for the Activity File using the parameters passed to it and connects it to the current User Log record (which, in its turn, is connected to the User File record for the current user). Here is a typical example:

```
Create Activity record
————————————————————
Parameter MAIN_FILE (Character   255)
Parameter RSN (Short number   0 dp)
Parameter TRANS_TYPE (Short integer   (0 to 255))
Begin reversible block
        Set main file {Activity file}
        Set read/write files {Activity file}
        Single file find on UL_SEQNO {#59}
End reversible block
Clear main file
Calculate AC_DATE_TIME as #D
Calculate AC_TYPE as TRANS_TYPE
Calculate AC_RECORD as con(MAIN_FILE,'/',RSN)
Prepare for insert with current values
Update files
```

The AC_RECORD value can be generated in a number of ways. In the above example the Main File name is retained as a string and concatenated with a delimiter and the RSN value. If a File number system were used, the value stored could be slightly shorter, but the fields maximum length, and therefore its index, would still have to be long enough to accommodate the largest possible file number and RSN.

We can shorten this further by storing the entire value as a decimal number, using the integer part to store either the File number or the RSN (your choice) and the decimal part to store the other. In either event, the field value will only require eight bytes and the index another eight. To protect numeric values placed in the decimal part that are an even multiple of ten, a nonzero dummy digit (usually a one) should be concatenated to the end before numeric conversion. The last digit must then be ignored when retrieving the data for reporting or record location purposes. The finished calculation statement using the numeric method would look like this (with AC_RECORD being defined as a floating point number):

```
Calculate AC_RECORD as con(MAIN_FILE,'.',RSN,1)*1
```

When reporting this information, the value of AC_RECORD must be parsed into it Main File and RSN components. If a numeric Main File indicator was used, it should be converted back to the name of the Main File using either an invisible, calculated automatic find field to retrieve the Data Dictionary record or a lookup channel to simply retrieve the value.

If you need to build a modification history list for a specific record, an exact match find will be required on the AC_RECORD field using a value generated in the identical manner you use for storage. This will need to be done in a *Repeat* loop, since a *Load connected records* command for the User Log File will have to be issued for each record located to read in the User information.
Other Examples

Here are abstracts of a few other possible uses for the Shadow Files concept for you to ponder. I have implemented variations of each of these in at least one application and have always gotten tremendously positive feedback as a result. Savvy clients *love* good statistics on their businesses!

Periodic Sales Tally

For a sales business that measures its success — as well as pays out commissions — on a monthly basis, summaries of sales were stored on an overall monthly record as well as in individual monthly records for each salesperson. This was further complicated by the fact that commissions were paid out on a sliding scale depending on total volume, but based upon a profitability figure associated with the product and the price paid rather than on the actual volume. Some sales were also shared among more than one person! Various profitability indices were derived from the overall monthly records and used for decision-making on future sales and ad campaigns. (An advertising effectiveness tracking system was also part of this application.)

Response Time Tracking

For the Work Request subsystem of an application for a large client, the status progress of Work Request records was monitored and summarized for each building (there are many hundreds of buildings!) on a monthly basis. Work Requests were distinguished as either "emergency" and "routine". The times to respond to the client, to call the appropriate contractor, and to the completion of the job were all compared to the time of the original request (received by telephone or email) and the results stored in "time range" fields for each stage (separate ranges for emergency and routine). Other related statistics were also updated in each record. These statistics could then be used for comparison with other months and other buildings in reports and charts or exported to spreadsheets for further analysis — even after the Work Request records had been archived and purged after a couple of years.

Inventory Movements

By tracking the effect of various transaction types in this application on inventory levels in a single File, I was able to generate a report that would yield the precise inventory for any date in the history of the datafile. What makes this more impressive is that there were a large number of possible transaction types, including Sale, Purchase, Delivery (outgoing), Shipment (incoming), and Borrow and Lend among other vendors. The inventory levels tracked were on hand, on order, on loan, borrowed, and spoken for (but not yet paid for or delivered)!

Revision Histories

For the Purchase Order subsystem of an application for a large client, changes to Purchase Order records were monitored. If the closing date or allocated amount of the PO were modified at the point in time where the operator clicked the OK button, the operator would be prompted for confirmation and the code of an authorizing supervisor. (Yes, their Activity was also being tracked!) A positive and complete response to this prompt led to the generation of a Change Order record connected to that PO record. A listing of Change Orders for the current PO record could be called up from the PO window. PO listings could be printed with or without Change Orders.

This overview of Shadow Files has, admittedly, been very brief and general — and I understand that you usually expect more specific details from me. Each implementation of this concept has its own unique character and challenges. If sufficient interest is shown in feedback to this article, I could expand on this subject further or create a three-day class to tour around. You can email me at dataguru@polymath-bus-sys.com.

David Swain.

David Swain, owner of Polymath Business Systems in Bedford, New Hampshire has been teaching OMNIS to eager students around the US since early 1985. His technical journal, OmniScience, remains a great source of detailed information on important features of OMNIS. You can learn more about these offerings through his Web site at http://www.polymath-bus-sys.com.

David Swain

Tips from the OMNIS Listserver

The following tips are from the OMNIS Listserver. To subscribe to the list, send a message to listserv@list.usa.net with no subject. In the message body, write “sub OU_DROP_POINT <your name>” without the quotes.

Auto Incrementing and OMNIS Data File

```
Test for only one user
If flag true
   If $cdata.$freesize<250000
      Yes/No message (Do you want to expand the
datafile ca 1 MB?)
                  If flag true
                  Calculate %DATAFILESIZE as
$cdata.$disksize Calculate
                  $cdata.$segments.1.$disksize as
%DATAFILESIZE+1000000 End If
                  End If
         End If
End If
```

Deleting Data Slots Without a Matching File Format Name

```
Set current list lSlotLst
Define list {lSlotName}
Calculate lFlag as
$cdata.$slots.$appendlist(lSlotLst,$ref.$name)
For each line in list from 1 to #LN step 1
Load from list
Calculate lFlag as $clib.$files.$findname([lSlotName])
If not(lFlag)&not(pos('#',lSlotName))     ;; Exclude system
slots
Calculate lFlag as $clib.$files.$add(lSlotName)
Delete data {[lSlotName]}
Calculate lFlag as
$clib.$files.$remove($clib.$files.[lSlotName])
End If
End For
Quit procedure
Local variable lFlag (Boolean)
Local variable lSlotLst (List)
Local variable lSlotName (Character   50)
```

This should be used with caution, it will delete any slot with no corresponding file format but the slot might be valid for another lib.

This code is also very useful for cleaning up a data file after you have been testing importing procedures. It is much faster to rename a slot than to delete it while you are running through the tests so this code will automatically wipe out the old test formats while you are doing something more constructive.

Library Name from a Window, File Format Name from a Field

```
Calculate MyLib as $winds.[#TOP].$lib().$name]
Calculate My_File as [FIELD_NAME].$format().$name>>
```

OMNIS

What's Up with Apple?

By Jordan Dea-Mattson,
Senior Evangelist,
Apple Computer, Inc.

For Apple watchers and those who do business with or in the world of Apple the last two months have been 'interesting', to say the least.

First, Gil Amelio resigns as CEO, then Steve Jobs steps into to try and get the company he co-founded back on track, and then - the biggest suprise of all - Apple and Microsoft announce a major deal that affirms their commitment to continue to deliver software for Mac OS.

Like a good soap opera - which is the role Apple plays for many in Silicon Valley - the story has unwound with more twists and turns and rumours, than truth and clarity, leaving many wondering, "What is going on with Apple and how is this going to affect my business?" In this article, I will lay out what has 'gone down' at Apple in the last two months and what it means for your business as OMNIS developers.

What happened with Gil?

In early July it was announced that Gil Amelio in consultation with Apple's Board of Directors had decided that the traits that Apple needed in a Chairman of the Board and Chief Executive Office (CEO) no longer made Dr. Amelio a good fit for the job.

Appropriately, Dr. Amelio resigned and the company is now involved in a search for a "world-class" CEO. The Board of Directors is looking for a strong, "customer focussed" CEO who will be able to maximise the value of Apple's product and technology portfolio.

The search for Apple's new CEO is being conducted by Fred Anderson, Apple's Chief Financial Office and acting CEO; Steve Jobs, Chairman and CEO of Pixar Animation Studios, Apple Co-Founder, and member of the Board of Directors; and Edgar S. Woolard, Jr., Chairman of the Board of Directors of E.I. duPont De Nemours and Co.

Dr. Amelio's strengths were in operations and strategic planning and he displayed his gifts in these areas in getting Apple onto the road to recovery. When he came to Apple, the company faced a host of crises. In particular, Apple faced a cash crisis of staggering proportions, major problems in product quality, and a lack of a clear and focussed product and operating system strategy. To his credit, Dr. Amelio addressed these problems and others during his tenure at Apple.

Steve's MacWorld Keynote

Given the anticipation with which Steve Jobs' MacWorld keynote speech was awaited, you would have thought that it was the second coming of Jesus Christ. In the days leading up to MacWorld Boston, speculation and rumours about what he would announce were rife. It was interesting, how often these rumors were inconsistent and contradictory. One source would announce that Steve was going to become CEO and Chairman of Apple Computer. Another would state with equal conviction that Apple was going to be broken up into separate hardware and software companies, and that the hardware company would be sold. Yet a third would announce that Apple was getting rid of the Mac OS and focusing solely on Rhapsody. The only thing missing in all of of this speculation and rumor-mongering was the truth.

Given the frenzied lead up to MacWorld, in some ways, Steve Jobs keynote was an anti-climax. Rather than focussing on selling the future to Apple's customers, he focussed on continuing the process of rebuilding Apple and selling what we have today to the customers that want to buy from us.

First, Steve announced a new Board of Directors for Apple that - in addition to Steve Jobs himself - added three top drawer business people with a depth and breadth of business experience. They were Bill Campbell, President and CEO of Intuit, who is also a former Apple Sales Executive and the CEO who built Claris into a major software business; Larry Ellison, President and CEO of Oracle; Jerry York, a top-drawer Chief Financial who is credited in stints at Chrysler and IBM with significantly adding these companies in their turnaround efforts. Steve Jobs, Bill Campbell, Larry Ellison, and Jerry York, joined from the previous Board of Directors, Edgar Woolard and Gareth Chang, Sr. Vice President of Hugh Electronics to make up Apple's new Board of Directors.

I am personally very excited about this Board, because all of the individuals involved are 'hands-on' and focused on getting results. They all have a reputation for focussing sharply on identifying customer needs and quickly addressing those needs.

Steve followed his announcement of Apple's new Board of Directors with what was the bomb-shell of the day, he announced a five point deal with Microsoft:

1. A broad patent cross-licensing agreement and balancing payments from Microsoft to Apple that resolved a long-running patent dispute between the two companies that had made it difficult for Apple and Microsoft to work together as closely as needed to fully support the Mac OS. While Apple and Microsoft have not disclosed the exact amount that Microsoft is paying in addition to the patent cross-licensing to solve this patent dispute, it is in the words of Fred Anderson, Apple's Chief Financial Officer, "Significant, but not material."

2. Microsoft will ship Microsoft Office, Internet Explorer, and other Microsoft product for Mac OS for a minimum of five years. This part of the deal is very important, because it means that your customers can count on the availability of Microsoft Office on the Mac OS for a minimum of five years. They don't have to worry about being left behind if they choose a Mac OS system.

3. Apple will continue to bundle Microsoft Internet Explorer with the Mac OS and make it the default browser in future releases of Mac OS. It should be noted that while Apple is bundling Internet Explorer and making it the default browser, Apple will continue to bundle Netscape Navigator and users will be able to easily switch between the two as their default browser.

4. Apple and Microsoft will collaborate to ensure compatibilitty between their respective Virtual Machines for Java and other programming languages. This means that Java as a platform will become a reality on Windows and Mac OS, allowing a Java program

written for one platform to run on the other without agressive tweaking and modification.

5. Microsoft will invest $150 million in non-voting Apple stock that it has to hold for a minimum of three years, at the end of which it can either sell the stock or convert it into voting shares. It is important to note, that even converted into voting shares, this investment gives Microsoft a less than 5% stake in Apple. It was suprising how many people thought that Microsoft had bought Apple for $150 million!

The big advantage of this investment is that when someone asks you if Apple is going to survive - a question that I get from everyone that finds out I am an Apple employee - you can honestly say, "Yes, because I don't think that the world's richest man is stupid or that he would throw away $150 million by investing it in a company that he thought wasn't going to survive."

Focusing on Mac OS...

During his keynote Steve focused on what he called Apple's biggest assets: its brand and the Mac OS. Long neglected and abused, Steve turned the spotlight on the Mac OS and promised that Apple would continue to invest and develop Mac OS going forward.

This focus was warranted in particular given that Mac OS 8 in its first two weeks in release sold through to customers over 1.2 millions copies! Many retail outlets in the US stated that they had not seen such strong sales of a software product since the introduction of Windows 95. In addition, the strong sales of Mac OS 8 has driven sales of Apple hardware at many resellers up by as much as 30-40%.

To paraphrase Steve Jobs, it will not be Tempo (Mac OS 8's code name), followed by Allegro (the code name for the next release of Mac OS), followed by Requiem. Apple will continue to develop and deliver high-quality upgrades of Mac OS to our customers.

Where is Rhapsody?

With Apple's renewed and strengten focus on Mac OS and in Steve's MacWorld keynote on selling what we have today, many wonder what is happening to Rhapsody. Simply put, Rhapsody is still happening. Apple continues to invest in Rhapsody and develop it.

Contrary to some rumours you may have heard, Apple has not cancelled Rhapsody or let go any of the Rhapsody team. In fact, the Rhapsody team is hiring and moving forward aggressively.

These rumours probably started because Steve didn't mention Rhapsody in his keynote. But he did this for a simple reason: he was focussed on what was happening today, what is available now. Too often in the past, Apple has sold a bright and shiny future at the expense of the present. This has been bad for Apple and our customers. By focusing on building a great Mac OS and selling to our customers today, while preparing Rhapsody for the future is good business.

Being Successful - selling what you have to those who want to buy it!

In essence, Apple is focusing on one thing: being successful. This means that we have to sell what we have - in hand, today - to those who want to buy from this. This means that we will be focusing on those markets where we are strongest, where we don't have to fight battles with customers to get them to buy from us and deliver. As Jean-Louis Gassee said to me once, "You must remember that customers are the ones that have your money in their pockets." Apple is now focused on building products today, for the customers that want to buy from us, so that we can get the money that is rightfully ours. For you as a member of the Mac OS developer community, this means that you can count on better and higher quality products that meet the needs of our common customers. This should be good for everyone: Apple, you, and our joint customers.

Jordan J. Dea-Mattson
Senior Evangelist
Tools & Utilities

Apple Computer, Inc.
1 Infinite Loop, MS: 303-2EV
Cupertino, CA 95014

Telephone: +1 (408) 974-4601
jordan@apple.com

SpellBinder Beta Available

You might be interested in checking out the beta 1 version of Spellbinder. This set of extended commands for Omnis provides the developer with a spell checking engine to check any data passed to it.
It supports multiple dictionaries and multiple languages.

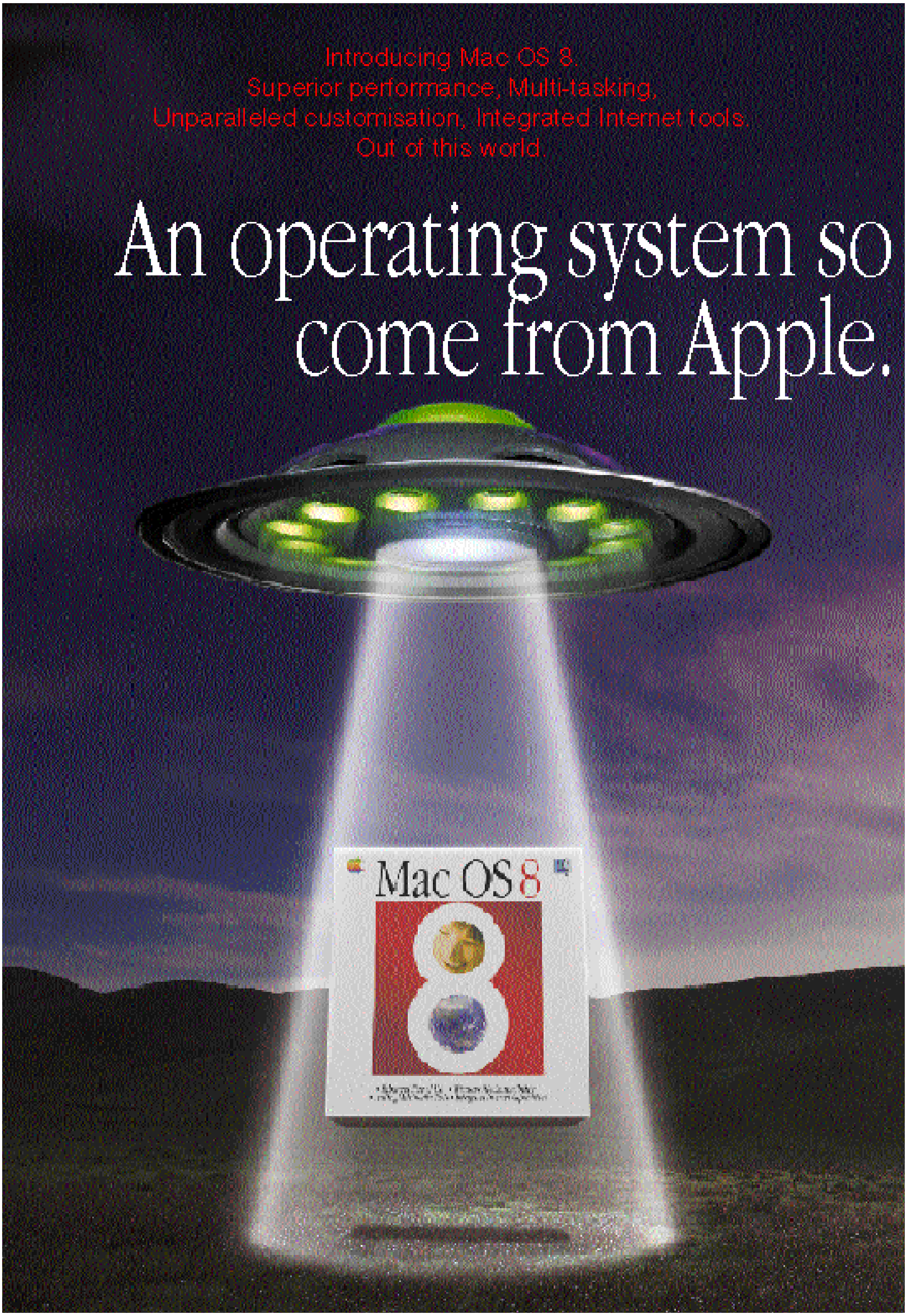
Beta 1 is available for Win 95, NT and PowerMac. Beta 2 will should include Mac68K and Beta 3 is Win 3.1.1.

You can access the files at: ftp://dynabyte.com/Pub/DynaByte/
PowerMac version - sbPowerMac.sea
Win95/NT version - sbWIN32.zip

I have included main dictionaries in all languages that I currently have. Additional languages are possible. Other specialty dictionaries will be included with the release. See the last page of the documentation for know problems, caveats and change histories.

On the sales side, we are offering special pre-release pricing. Contact me directly via email.

Jerry Lipscomb
jlipscomb@dynabyte.com
http://www.dynabyte.com


Introducing Mac OS 8.
Superior performance, Multi-tasking,
Unparalleled customisation, Integrated Internet tools.
Out of this world.
An operating system so
come from Apple.
Mac OS 8

advanced it could only
(Or did it?)

Get ready for a close encounter of the eighth kind. Mac® OS 8 is here. And with it comes a whole new way of working with your Mac OS computer.

Mac OS 8's purpose on this planet is to help you work more productively, and remove the barriers between you and your full potential. And it does this with performance that's out of this world. Thanks to the Mac OS 8 improved multi-tasking Finder™ software, you can accomplish several tasks at once, like opening and copying files without any waiting around. But all this power is nothing without organisation. That's why its new unparalleled customisation features let you adapt your system to match your individual work style. Spring-open folders, contextual menus and pop-up windows help get you around faster, and are part of a new 3-dimensional look that's even easier to work with. And Mac OS 8 has integrated Internet capabilities. Internet Explorer, Netscape Navigator,™ the PointCast Network and a new Internet Setup Assistant are built right in, making it easier than ever to get on the Net, to email, surf or browse the cosmos.* All this lets you work the way you think, so you can be more efficient. It even includes QuickTime® multimedia technology. And to show that Mac OS 8 comes in peace, it can read Windows files, whether you have the original application or not. It's also compatible with your existing Mac applications. Mac OS 8 is designed for 68040 and PowerPC™-based systems, and features reliability improvements that'll have you glad you went with the power of 8.

Intelligent life is out there. Mac OS 8 is proof.

For your own close encounter with Mac OS 8, visit our Web site at http://www.apple.com.au for more information and the names and addresses of your local resellers. Or give us a ring at 1-800 025 355.

MacOS

What a difference 8 makes.

Project X

This is the first article of three that will explain and clearly define the entire 'Project X' concept, and include examples of how to get it all done. These articles will also contain code samples that show a good starting point for getting the job done.

Project X is all about traversing an Omnis Library file's notation tree. The idea is that because Omnis is composed entirely of objects and attributes, it should be possible to reduce a library to text.

It was with this knowledge in mind that made the idea of extracting an entire Omnis library file into a text document feasable. By the same token, it is possible to turn a text document (rigourously formatted, of course) into an Omnis library.

Hence, Project X has 3 components:

1 - Reduce an Omnis Library to a text document
2 - Create an Omnis Library from another via notation
3 - Rebuild an Omnis Library from text

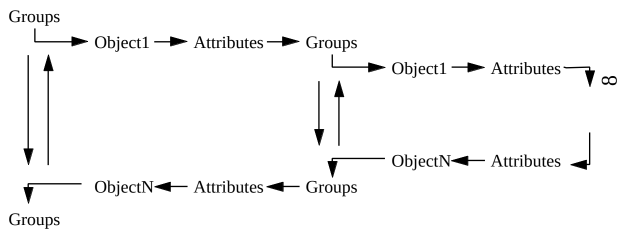
Why Project X?

Why copy Omnis files in this manner? Isn't iy easier (and faster) to simply copy formats through the format browser? What advantages can be gained from reducing Omnis to Text?

When corruption occurs at token level in Omnis, it's game over. That format needs to be taken from a backup, or rewritten. Project X will remove the Omnis commands and attributes from a tokenized environment. Any corruption that has occured has been wiped out completely.

In order to accomplish Project X using the generic algorithm provided, Omnis will alter the names of Format Variables during processing, successfully perform the algorithm recursively, and interrogate an Omnis Library file.

All of these goals are accomplished with no fuss or issues from Omnis, and without the use of any special externals or functions. This only serves to re-enforce what a powerful development tool Omnis is by itself.



By using recursion, an Omnis library can traverse the Notation Tree as above, stopping and exploring every branch. Always falling back successfully to the previous level to continue processing each group.

Code & Explanation

The entire code is stored in a menu , activated on the first line.

```
0  TextBuilder
```

```
;  This small menu takes apart a library notationally.
;  Simply install it in a library and run it.
;  The data will be placed in a text file.
```

The 'Begin' Procedure renames the library stripper application to 'NOTELIB' and closes all other running libraries. It will then request an open library which it will name 'source'. The source library will be interrogated.

```
1  Begin!
Calculate #F as $clib.$name.$assign('NOTELIB')

Set current list LV_LibList
Define list {LV_LibName}
Calculate LV_LibList as
$root.$libs.$makelist($ref().$name)
Redefine list {LV_LibName}
If #LN>1
  For each line in list from 1 to #LN step 1
    Load from list
    If
LV_LibName<>'NOTELIB'&LV_LibName<>'O7DSHELL'
      Close library {[LV_LibName]}
    End If
  End For
End If

Prompt for library (Do not close other libraries,Do not call
startup procedure) {source}
If flag false
  Quit procedure
End If
Calculate $ignoreExternal as kTrue

Call procedure 9 ('$root.$libs.source') {CallProg}

Local variable LV_LibList (List)
Local variable LV_LibName (Character   10000000)
```

During the course of the processing, Format Variables are renamed countless times. This simple procedure resets them to their default names. It is run once at the beginning and once at the end of the process.

```
8  Rename FVs
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.12.$name.$assign('FV1')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.13.$name.$assign('FV2')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.14.$name.$assign('FV3')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.15.$name.$assign('FV4')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.16.$name.$assign('FV5')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.17.$name.$assign('FV6')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.18.$name.$assign('FV7')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.19.$name.$assign('FV8')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.20.$name.$assign('FV9')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.21.$name.$assign('FV10')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.22.$name.$assign('FV11')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.23.$name.$assign('FV12')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.24.$name.$assign('FV13')
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.25.$name.$assign('FV14')
```

```
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.26.$name.$assign('FV15')
```

Procedure 9 will set up the text file the export will be made to in the directoy that the 'source' library is stored, it sets the depth of the recursion to 0, and closes the source application once everything is complete.

```
9  CallProg
Parameter P_PathName (Character   10000000)
Calculate LV_PathName as sys(10)
SplitPathname2 (LV_PathName,LV_Drive,LV_Dir)
Set print file name {[con(LV_Drive,LV_Dir,'Result.txt')]}

Calculate FV_FirstPath as P_PathName

Calculate FV_Depth as 0

Call procedure 8 {Rename FVs}

Call procedure 11 (P_PathName) {BeginTree}

Close library {source}

Close print file

Call procedure 8 {Rename FVs}

Quit procedure
Local variable LV_PathName (Character   10000000)
Local variable LV_Drive (Character   10000000)
Local variable LV_Dir (Character   10000000)
```

Procedure 10 holds all the Format Variables.

```
10  FormatVariables
Format variable FV_PATHTOFOLLOW (Character
10000000)

Format variable FV_Counter (Character   10000000)
Format variable FV_List (List)

Format variable FV_Path (Character   10000000)
Format variable FV_Group (Long integer)
Format variable FV_Value (Character   10000000)
Format variable FV_CanAssign (Boolean)

Format variable FV_ListPath (Character   10000000)
Format variable FV_att_List (Character   10000000)
Format variable FV_obj_name (Character   50)

Format variable FV_FirstPath (Character   10000000)
Format variable FV_Depth (Short integer   (0 to 255))

Format variable FV1 (List)
Format variable FV2 (List)
Format variable FV3 (List)
Format variable FV4 (List)
Format variable FV5 (List)
Format variable FV6 (List)
Format variable FV7 (List)
Format variable FV8 (List)
Format variable FV9 (List)
Format variable FV10 (List)
Format variable FV11 (List)
Format variable FV12 (List)
Format variable FV13 (List)
Format variable FV14 (List)
Format variable FV15 (List)
```

Procedure 11 is the recursed procedure, and it is where all the processing of the branches takes place. It begins by assigning the current branch of the notation tree to a corresponding Format Variable depending on the depth of the recursion (ie FV1 is the first occurrence of the app and the last one to close). It exports all the assignable information to a text file, and then interrogates for groups within that branch.

If a group is discovered, the procedure is recursed, the depth is increased, and a new format variable is assigned. Any groups within that group will then be interrogated and so on.

After the groups have been interrogated, or if no groups have been found, the procedure closes and falls back to the previous occurrence, where processing continues.

The depth will rise and fall regularily, but the first occurrence is always the first one to begin and the last one to finish.

Notice that I have explicitly excluded $add from the interrogation of the attributes. If $add is not excluded, an object will be added 3 times as the procedure steps past $add, $addbefore, $addafter!

```
11  BeginTree
Parameter P_PathName (Character   10000000)
Other parameters are optional
Parameter P_GetMembers (Long integer) = 0

Call procedure 16 (P_PathName) {Dropout}
If flag false
  Quit procedure
End If

Calculate FV_Depth as FV_Depth+1

Call procedure 12 (P_PathName) {StripString} with return
value FV_ListPath
Call procedure 13 (FV_ListPath)
{CreateListFormatVariable}
Calculate FV_ListPath as con('FV',FV_Depth,FV_ListPath)

Redraw working message
Working message (High position,Large size) {[FV_Depth]
- [P_PathName]}
Set current list [FV_ListPath]
Define list (Store long data)
{FV_Path,FV_Group,FV_Value,FV_CanAssign,FV_att_List}

Calculate FV_Counter as 1
Transmit text to print file (Add newline)
Transmit text to print file (Add newline) {[P_PathName]}
```

George Schwalbe

```
Repeat
  Calculate LV_Position as
pos('$add',con([P_PathName].$att(FV_Counter).$name))
  If LV_Position=0
    Calculate FV_att_List as
    Calculate FV_Path as
[P_PathName].$att(FV_Counter).$name
    Calculate FV_Group as
[P_PathName].[FV_Path].$count
    Calculate FV_Value as [P_PathName].[FV_Path]
    Calculate FV_CanAssign as
[P_PathName].[FV_Path].$canassign
    If FV_Group>0
      Calculate FV_att_List as
con(P_PathName,'.',FV_Path)
      Transmit text to print file (Add newline)
{[FV_Path],[FV_Group],[FV_Value],[FV_CanAssign],[FV_att_List]}
    Else
      If FV_CanAssign=kTrue
        Transmit text to print file (Add newline)
{[FV_Path],[FV_Group],[FV_Value],[FV_CanAssign],No
List}
      End If
    End If
    Add line to list
  End If
  Calculate FV_Counter as FV_Counter+1
Until FV_Counter>[P_PathName].$attcount()

If P_GetMembers>0
  Calculate [#CLIST] as
[P_PathName].$makelist($ref().$fullname,$ref().$attcount)
  Redefine list {FV_att_List,FV_Group}
End If

Calculate #L as 1
Repeat
  Load from list
  Calculate LV_Position as pos('$',FV_att_List)
  If LV_Position=0
    Calculate FV_att_List as
con(P_PathName,'.',FV_att_List)
  End If
  If FV_Group
    Call procedure 11
(FV_att_List,[con(FV_att_List,'.$count')]) {BeginTree}
  End If
  Calculate #L as #L+1
Until #L>#LN

Call procedure 15 (FV_ListPath)
{RemoveFormatVariables}
Calculate FV_Depth as FV_Depth-1
Call procedure 14 (FV_ListPath) {GoBackPath} with return
value FV_ListPath
Calculate #F as
$root.$libs.NOTELIB.$menus.NoteMenu.$fvardefs.[FV_ListPath].$ident
If flag false
  Call procedure 14 (FV_ListPath) {GoBackPath} with
return value FV_ListPath
End If
If FV_ListPath<>'FV0@root@libs'
  Set current list [FV_ListPath]
End If
Quit procedure

Local variable LV_Position (Long integer)
```

Procedure 12 strips the '$' and any other characters from the notation strings, and creates a string that Omnis will accept as a format variable name.

```
12  StripString
Parameter P_TO_STRIP (Character   10000000)

Repeat
  Calculate LV_Counter as pos('$',P_TO_STRIP)
  If LV_Counter>0
    Calculate LV_Before as
mid(P_TO_STRIP,1,LV_Counter-2)
    Calculate LV_After as
mid(P_TO_STRIP,(LV_Counter+1),1000)
    Calculate P_TO_STRIP as con(LV_Before,'@',LV_After)
  End If
Until LV_Counter=0

Repeat
  Calculate LV_Counter as pos('.',P_TO_STRIP)
  If LV_Counter>0
    Calculate LV_Before as
mid(P_TO_STRIP,1,LV_Counter-1)
    Calculate LV_After as
mid(P_TO_STRIP,(LV_Counter+1),1000)
    Calculate P_TO_STRIP as con(LV_Before,'!',LV_After)
  End If
Until LV_Counter=0

Repeat
  Calculate LV_Counter as pos('//',P_TO_STRIP)
  If LV_Counter>0
    Calculate LV_Before as
mid(P_TO_STRIP,1,LV_Counter-1)
    Calculate LV_After as
mid(P_TO_STRIP,(LV_Counter+2),1000)
    Calculate P_TO_STRIP as con(LV_Before,'',LV_After)
  End If
Until LV_Counter=0

Set return value {P_TO_STRIP}
Quit procedure

Local variable LV_Counter (Long integer)
Local variable LV_Before (Character   10000000)
Local variable LV_After (Character   10000000)
```

Procedure 13 renames the format variable. If a name cannot be assigned, the program dies.

```
13  CreateListFormatVariable
Parameter Listname (Character   10000000)

Calculate #F as
$root.$libs.NOTELIB.$formats.NoteMenu.$fvardefs.[con(FV,FV_Depth)].$name.$assign(con(FV,FV_Depth,Listname))
If #F=0
  OK message {Critical Error Occured At [FV_Depth] -
[Listname]}
  Quit all procedures
End If
```

Procedure 14 strips the last name from the current list, and reassigns the current list to the one that is currently being processed on the previous occurrence of the procedure.

```
14  GoBackPath
;  64 is the ASCII Value of '@'
;  33 is the ASCII value of '!'
Parameter P_Go_Back (Character   10000000)

Calculate L_Position as len(P_Go_Back)
Calculate L_Done as kFalse

Repeat
  If
asc(P_Go_Back,L_Position)=64|asc(P_Go_Back,L_Position)=33
    Calculate L_Done as kTrue
    Calculate L_Position as L_Position-1
    Calculate P_Go_Back as mid(P_Go_Back,1,L_Position)
  Else
```

```
   Calculate L_Position as L_Position-1
  End If
Until L_Done=kTrue

Calculate L_Position as pos('@',P_Go_Back)
Calculate P_Go_Back as mid(P_Go_Back,L_Position,1000)
Calculate P_Go_Back as con('FV',FV_Depth,P_Go_Back)

Set return value {P_Go_Back}
Quit procedure
```

Procedure 15 renames the format variable to its original name.

```
15  RemoveFormatVariables
Parameter Listname (Character   10000000)

Calculate #F as
$root.$libs.NOTELIB.$formats.NoteMenu.$fvardefs.[Listname].$name.$assign(con('FV',FV_Depth))
If #F=0
  OK message {Critical Error Occured At [FV_Depth] -
[Listname]}
  Quit all procedures
End If

Quit procedure
```

Procedure 16 will allow you to avoid specific areas. I have avoided the 'Formats' area due to the fact that every single format in the application resides there, as well as the system formats.

```
16  Dropout
Parameter P_Pathname (Character   10000000)

If P_Pathname='$root.$libs.source.$formats'
  Quit procedure (flag clear)
End If
```

There we are, how to traverse the notation tree in several easy steps. Of course, an application attempting a task of this nature is no fireball. This is only a starting point. Faster and more robust versions have been developed since this magazine has gone to print. I would appreciate any feedback or suggestions anyone may have.

George Schwalbe
george_schwalbe@dlagroup.com.au.

OMNIS 7 v3.6 Patch

The OMNIS R&D team now has a patch that addresses most, if not all, of the issues that have been raised so far with OMNIS 7 3.6 on the Apple
platforms. This patch will be made available next week to selected customers for verification of the fixed faults.

The release schedule, and information regarding the "general" availability of this patch will be posted on the technical support section of the OMNIS web site
(www.omnis-software.com) as soon as this information becomes available.

Negin Hakimian
Director, Worldwide Support
OMNIS Software
851 Traeger Avenue
San Bruno, CA 94066
http://www.omnis-software.com

Uppy.lbr. Towards a smaller Upgrade library.

Uppy is a little demo library that can be used to create a subset of you library for Upgrade purposes. Instead of sending your clients a full new version why not just send them the differences? The saving in space requirements, transmission costs can be quite substantial.

This library demonstrates a couple of different processes:

1: Building you 'read.me' files into your 'About Window'.
2: Comparing the equivalency of formats.
3: Replacing STARTUP menus of libaries on the fly and other 'tricky bits'.
4: Lots of notation examples.

This demo program follows a standard structure that I have adopted for all demonstration libraries:
The STARTUP code closes all data files and other libraries, checks for externals and sets some preferences.

```
Calculate $root.$prefs.$helpbaron as kFalse
Calculate $clib().$name as 'Uppy'; renames the internal name to Uppy
;close all other libraries
Set current list List_Libs
Define list {Lib_Names}
Calculate List_Libs as $root.$libs.$makelist($ref().$name); could have used
$appendlist
Redefine list {Lib_Names}
Calculate #L as 1
Repeat
          Load from list
                    If Lib_Names<>'Uppy'
                              Close library {[Lib_Names]}
                    End If
          Calculate #L as #L+1
Until #L>#LN
Close data file ;close all data files
Calculate $clib.$ignoreExternal as kTrue; enable reading of other libraries
OK message (Large size) {All other libraries have been closed and all data
files have been closed}
;check for the externals used
Set current list List_Externals
Define list {External_Function_Name}
Build externals list
Set search as calculation {External_Function_Name='ENV00202'}
Search list (From start)
If flag false
          OK message {The ENV00202 externals must be in your externals
directory}
          Quit procedure
End If
```

The 'About Window'

This window is opened with a call to this procedure contained within the About Window:

```
Call procedure 185 {VersionTextGrinder}
Open window [$cformat.$name]/CENTER
```

Before opening the window the procedure text is read and converted to a 'displayable form' for placement in the on screen display/edit window:

(Procedure 186 contains $proctext like the following:

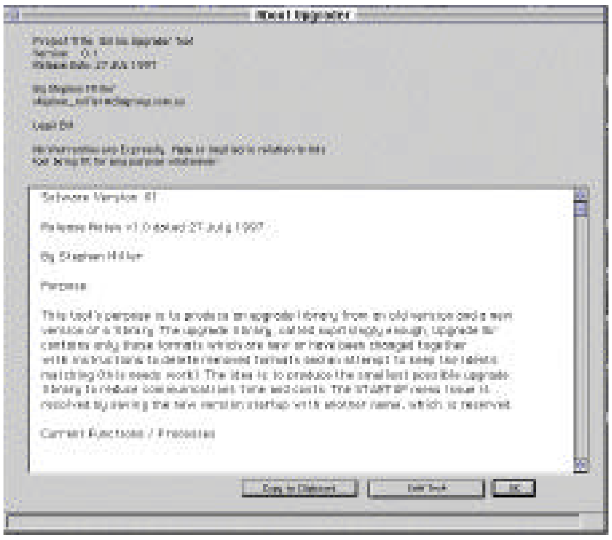
```
;  Sotware Version .01
;
;  Release Notes v1.0 dated 27 July 1997  etc

Procedure 185 (Version Text Grinder) is as follows:

Calculate TextRaw as $cformat.$procs.186.$ref().$proctext
While pos(';',TextRaw)
          Calculate Pos_Counter as pos(';',TextRaw)
          Calculate Text_First as mid(TextRaw,1,(Pos_Counter-1))
          Calculate Text_Second as mid(TextRaw,(Pos_Counter+1),5000)
          Calculate TextRaw as con(Text_First,Text_Second)
End While
```

```
Calculate fvRelease_Text as TextRaw (Redraw field)
```

the 'About Window' opens:



About Upgrader

Project Title: Omnis Upgrader Tool
Version: 0.1
Release Date: 27 JUL 1997

By Stephen Hillier
[illegible]

Legal Bit

[illegible]

Software Version: [illegible]

Release Notes v1.0 dated 27 July 1997

By Stephen Hillier

Purpose

This tool's purpose is to produce an upgrade library from an old version and a new version of a library. The upgrade library, called surprisingly enough, Upgrade.lbr contains only those formats which are new or have been changed together with instructions to delete removed formats and an attempt to keep [illegible] matching (this needs work!) The idea is to produce the smallest possible upgrade library to reduce communications time and costs. The STARTUP name issue is resolved by saving the new version startup with another name, which is reserved.

Current Functions / Processes

The Proctext is visible in the scrollable text field. You can copy the text to the clipboard or edit it.

To Edit:

```
Calculate Edit_Text as fvRelease_Text (Redraw field)
Hide fields 1090 to 1090; note the use of the object number.
Show fields 1094 to 1094
Show fields 1096 to 1096
Enter data
If flag true
        Call procedure 14 (Edit_Text) {Prepare string}
        (Procedure 14 appears immediately below)
        Parameter Text_In (Character   10000000)
        Calculate %%TEXT as Text_In
        While pos('  ',Text_In)
                Calculate %POS as pos('  ',Text_In)
                Calculate %%TEXT_ONE as mid(Text_In,1,%POS-1)
                Calculate %%TEXT_TWO as
md(Text_In,%POS+2,len(Text_In))
                Calculate Text_In as con(%%TEXT_ONE,'
',%%TEXT_TWO)
        End While
        While pos(';',Text_In)
                Calculate %POS as pos('  ',Text_In)
                Calculate %%TEXT_ONE as mid(Text_In,1,%POS-1)
                Calculate %%TEXT_TWO as
mid(Text_In,%POS+1,len(Text_In)) Calculate Text_In as con(%%TEXT_ONE,'
',%%TEXT_TWO)
        End While
        Calculate Text_In as con(';',Text_In)
        While pos(chr(13),Text_In)
                Calculate %POS as pos(chr(13),Text_In)
                Calculate %%TEXT_ONE as mid(Text_In,1,%POS-1)
                Calculate %%TEXT_TWO as
mid(Text_In,%POS+1,len(Text_In)) Calculate Parse_Front as
con(Parse_Front,%%TEXT_ONE,chr(13))
                Calculate Text_In as con(';',%%TEXT_TWO)
        End While
        If len(Text_In)
                Calculate Parse_Front as  n(Parse_Front,Text_In,chr(13))
        End If
         #F as $cformat.$procs.186.$proctext.$assign(Parse_Front)
        Save format {[$cformat.$name]}
        Local variable Parse_Front (Character   10000000)
        (end insert of called procedure).
        Call procedure 185 {VersionTextGrinder}
        Hide fields etc.
End If
```

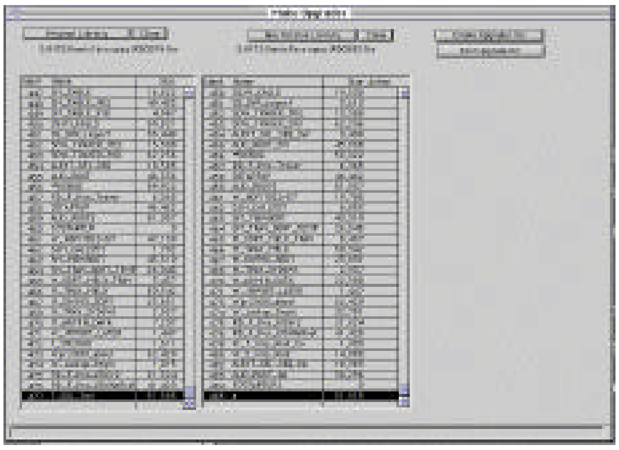
The above approach encapsulates the 'read.me' text and is reusable from one product to another. Enhancements would include a 'Show Read Me Button' that would alternatively hide and show the Read me text field and option Buttons.

The Upgrader Window

This windows the opening or the Original Library and the New Version of the Original. The Libraries are compared and a Third Library. 'Upgrade.lbr' is created which contains the new or modified formats, a special STARTUP format and the new version of the STARTUP format stored under another name.

To Open and compare the two libraries.
(Code shown opens the first library and calculates the checksum).

```
If len(Original_name)
        Close library {Original}
End If
Prompt for library (Do not close other libraries,Do not call startup procedure)
{Original}
If flag true
        Set current list list_current
        Define list {format_name,Format_Ident,format_size}
        Calculate Original_name as $root.$libs.Original.$pathname (Redraw
field)
        Calculate list_current as
$root.$libs.Original.$formats.$makelist($ref().$name,$ref.$ident,'')
        Redefine list {format_name,Format_Ident,format_size}
        Calculate #L as 1
        Repeat
                Load from list
                Calculate BIN_FORMAT as
$root.$libs.Original.$formats.[format_name].$formatdata
                Calculate format_size as ENV002C(BIN_FORMAT)
                Replace line in list
                Calculate #L as #L+1
        Until #L>#LN
        Set sort field Format_Ident
        Sort list
        Clear sort fields
Else
        Set current list list_current
        Define list {format_name,format_size,format_type}
End If
Redraw lists
```



The following code creates the UPGRADE.LBR:

```
Call procedure 30 {Compute Deleted}
Call procedure 33 {Compute Insert and Modified}
Call procedure 31 {Compute Insert and Delete}
Call procedure 32 {Create Startup Menu}
(The Create Startup Menu code is as follows)
Working message (High position,Large size) {Creating Upgrade.lbr Startup
menu. Note your screen may appear to freeze  for a few moments.....}
Calculate Total_Proc as ''
Set current list list_new
Calculate #L as 1
Repeat
        Load from list
        If format_size>0     ;; bad format control
          If REASON='N'
          Calculate ProcTextAdd as con('Calculate #F as$root.$libs.
Object.$formats.$add("',format_type,'","',format_name,'")',chr(13))
        Calculate ProcTextAdd as con('Calculate #F as $root.$libs.Object.
        $formats.',format_name,'.$formatdata.$assign($root.$libs.
        Upgrader.$formats.',format_name, '.$formatdata)',chr(13))
                Else If REASON='I'
                        Calculate ProcTextAdd as con('Calculate #F as
$root.$libs.Object.$formats.$add("',format_type,'","',format_name,'")',chr(13))
                Calculate Format_Group as con('$',format_type,'s')
```

```
                                Calculate ProcTextAdd as con('Calculate #F as

$root.$libs.Object.',Format_Group,".$remove ('",format_name,"')",chr(13))
                                Else If REASON='C'
                                  If format_name='STARTUP'
                                  Calculate ProcTextAdd as con('Calculate #F as
                                 root.$libs.Object.$formats.STARTUP.$formatdata.$assign
                                ($root.$libs.Upgrader.$formats.','XXXXSTARTUP','.
$formatdata)',chr(13))
                                Else
                                Calculate ProcTextAdd as con('Calculate #F as
$root.$libs.Object.$formats.',format_name,'.$formatdata.$assign($root.
                $libs.Upgrader.$formats.',format_name,'.$formatdata)',chr(13))
                                                End If
                                Else If REASON='D'
                                Calculate Format_Group as con('$',format_type,'s')
                                                Calculate ProcTextAdd as con('Calculate #F as

$root.$libs.Object.',Format_Group,".$remove ('", format_name,"')",chr(13))
                                End If
                                Calculate Total_Proc as con(Total_Proc,ProcTextAdd)
                End If
                Calculate #L as #L+1
Until #L>#LN
Calculate #F as
$root.$libs.Uppy.$menus.sysm_UPSTART.$procs.1.$proctext.$assign(Total_Proc)
Close working message
Quit procedure
(Create Startup Menu Code ends)


Working message (High position,Large size) {A New Library called Upgrade.lbr
has been created and the formats are being copied......}
                Calculate Path as sys(10)
                SplitPathname2 (Path,'',Drive,'','')
                Create library (Do not close other libraries)
                {[con(Drive,'Upgrader.lbr')],Upgrader}
                Set current list list_new
                Calculate #L as 1
Repeat
                Load from list
                If format_size>0      ;; bad format control
                                If format_name='STARTUP'
                                                Calculate #F as
$root.$libs.Upgrader.$formats.$add(format_type,'XXXXSTARTUP')
                                                Calculate #F as
$root.$libs.Upgrader.$formats.XXXXSTARTUP.
                $formatdata.$assign($root.$libs.NewVersion.
$formats.STARTUP.$formatdata)
                                                Save format {Upgrader.XXXXSTARTUP}
                                Else
                                                Calculate #F as
$root.$libs.Upgrader.$formats.$add(format_type,format_name)
                                                Calculate #F as
$root.$libs.Upgrader.$formats.[format_name].$formatdata.
                                $assign($root.$libs.NewVersion.
$formats.[format_name].$formatdata)
                                                Save format {[con('Upgrader.',format_name)]}
                                End If
                End If
                Calculate #L as #L+1
Until #L>#LN
Calculate #F as $root.$libs.Upgrader.$formats.$add(format_type,'STARTUP')
Calculate #F as $root.$libs.Upgrader.$formats.STARTUP.$formatdata.
$assign($root.$libs.Uppy.$formats.sysm_UPSTART.$formatdata)
Close working message
Calculate UpgraderPath as $root.$libs.Upgrader.$pathname
Close library {Upgrader}
OK message (High position,Large size) {[UpgraderPath] has been created and
closed}
```

A menu format called sysm_UPSTART is copied into the Upgrader.lbr and renamed as STARTUP.

The following code that will execute when you open the Upgrade.lbr and point it at the library you wish to upgrade.

```
Other parameters are optional
Parameter MODE (Character   10000000)
Calculate $root.$prefs.$helpbaron as kFalse
Calculate $clib().$name as 'Upgrader'
Set current list List_Libs
Define list {Lib_Names}
Calculate List_Libs as $root.$libs.$makelist($ref().$name)
Redefine list {Lib_Names}
Calculate #L as 1
Repeat
                Load from list
                If Lib_Names<>'Uppy'&Lib_Names<>'Upgrader'&MODE='Test'
                                Close library {[Lib_Names]}
                Else If Lib_Names<>'Upgrader'&MODE=''
                                Close library {[Lib_Names]}
                End If
                Calculate #L as #L+1
Until #L>#LN
Close data file
Calculate $clib.$ignoreExternal as kTrue
OK message (Large size) {All other libraries have been closed and all data
files have been closed}
Prompt for library (Do not close other libraries,Do not call startup procedure)
{Object}
Working message {Copy Formats in Upgrade}
                Call procedure 1; The notation string that does the copying of
formats
Close working message
OK message {Upgrader complete. The library you have selected will now be
opened.}
Calculate PathObject as $root.$libs.Object.$pathname
Close library {Object}
Open library {[PathObject]}
```

Once the library you wish to Upgrade has been opened the above code calls the notation to do the copying in Procedure slot 1, which may look like the following:

```
Calculate #F as $root.$libs.Object.$formats.
F_Imp_DESTandOVN2.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_DESTandOVN2.$formatdata)
F_Destn.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Destn.$formatdata)
F_Imp_DESTandOVN.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_DESTandOVN.$formatdata)
F_Imp_Destctl.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_Destctl.$formatdata)
F_Imp_Hdr.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_Hdr.$formatdata)
F_Imp_INT_CODES.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_INT_CODES.$formatdata)
F_Imp_Rout_Con.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_Rout_Con.$formatdata)
F_Imp_Trnks.$formatdata.$assign($root.$libs.Upgrader.$formats.F_Imp_Trnks.$formatdata)
```

This code would include a call similar to the above to copy the correct correct $formatdata into the STARTUP menu:

```
STARTUP.$formatdata.$assign($root.$libs.Upgrader.$formats.XXXXSTARTUP.$formatdata)
```

(Note there are easier ways of doing this, code similar to the code that compares the two libraries initially could have been used instead of the above).

It would be possible to modify this code so that it is activated automatically by a version check outlined in a recent article by Jim Pistrang in the previous edition of OmniDirectional. The method to upgrade the object library procedurally, the above method assumes an manual upgrade would involve passing parameters when the object library is opened as follows:
Open library {Mylibrary,object,MyPassword, ('DoUpgrade')}

'Uppy' can be downloaded from the DLA FTP site, ftp://ftp.dlagroup.com.au/pub/Omnis/Downloads/General/uppy.lbr. This location is also accessible via the DLA web page, in the resources section. Enjoy…

Stephen Miller
stephen_miller@dlagroup.com.au

Stephen Miller with Dean Sappey of Perfect Technology at DLA's Launch of OMNIS Stufdio

The Do Commands and How to Use Them

By Leon Venter

OMNIS Studio has a family of six commands, namely the Do commands, for invoking methods. Although these commands might appear to be similar, each one has unique capabilities and a specific role. They can be split into three groups:

A) Do, Do method, Do code method
B) Do inherited, Do default
C) Do redirect

The commands in group A are the most general in nature and the most commonly used. They allow you to explicitly specify the method to be executed and the parameters to be passed. The 'Do' command is used to invoke instance methods. The 'Do method' command is used to call methods within the same instance or class. The 'Do code method' command allows you to invoke class methods.

The commands in groups B and C are more specialized and more limited. In their case, the method to be invoked and the parameters to be passed are implicit. The 'Do inherited' command is used to invoke the base-class version of a method that you've overriden in a subclass. Similarly, the 'Do default' command is used to invoke the standard behavior of a notational attribute or function that you've overriden. The 'Do redirect' command allows you to set up custom and/or dynamic event-handling chains in your applications.

Do

Typical syntax: Do <instance>.<publicmethodname> (parms...) returns [resultvar]

<instance>

A reference to an instance. It can be specified using:

1) notation, e.g. $topwind, $ctask, $imenus.MyMenu, etc
2) an Item reference variable, or
3) an Object variable.

To specify the current instance, you must use $cinst, e.g. Do $cinst.$redraw(). (Unlike C++, there is no implicit 'this' pointer.)

<publicmethodname>

A method name that begins with '$' or a notational function name.

[resultvar]

An optional variable. You can set a reference (as with the Set reference command) by specifying an Item reference-type variable as the resultvar.

The 'Do' command is essentially equivalent to the Calculate command, i.e. it evaluates an expression and then places the result in a return field, if specified. For example:

```
Do $topwind.$insert();; Invokes the $insert method of the
top window instance
Do $cinst.$double(4) + 5  returns myNumber;; Passes 4
to the $double method of the current instance, adds 5 to
the return value and stores the result in myNumber
Do 'Hello world'  returns myString   ;; Assigns the
specified value to myString
```

'Do' is the primary command for specifying the interactions between all instances in an OMNIS application. None of the other Do commands will do! (The less common means of specifying instance interactions are via the Calculate command and square bracket notation).

Note that the 'Do' command can only be used to invoke the public methods of an instance. It cannot be used to call private instance methods, or methods in code classes.

To allow your classes to be sub-classed, you should ALWAYS use 'Do' when calling PUBLIC methods in the current instance (i.e. do not use 'Do method'). This is because 'Do' is the only one of the Do commands that supports polymorphism, since it does virtual method dispatching, i.e. run-time binding. (See Do method for more details).

By the way, it is important (and useful) to note that public instance methods are invoked in exactly the same context as built-in OMNIS functions. This means that you can write your own functions as public methods of an object class and use them in any calculation.

For example, if you need a function that returns the last n characters of a string,, you could, of course, just use the mid() function, e.g. Calculate myResult as mid(myString, len(myString) - n + 1, n). However, if you need to do this in more than one place in your application, you could make life easier by creating an object class called 'MyFuncs', with a public method called '$right', which looks like this:

```
        Parameters:   pString  (char)
                      pLen  (long number)
        Quit method mid(pString, len(pString) - pLen + 1,
pLen)
```

To use this function in your application, create an appropriately-scoped variable (i.e. an instance, class or task var) named gFuncs, with type = 'Object' and subtype = 'MyLib.MyFuncs'. Then use it in any calculation expression, as follows:

```
Calculate myResult as con( myPrefix,
gFuncs.$right(myString, n) )
```

or

```
OK message {The last four digits of your phone number
are: [gFuncs.$right(Cust_Phone, 4)]}
```

Do method

Syntax: Do method <methodname> (parms...) returns [resultvar]

<methodname>

A method in the same class as the calling method.

[resultvar]

An optional variable. You can set a reference (as with the Set reference command) by specifying an Item reference-type variable as the resultvar.

The 'Do method' command is similar to the OMNIS Classic command 'Call procedure <procedurename>'.

It is used to call methods within the same class. Note that, if the calling method belongs to an object on a window (e.g. an entry field), OMNIS will first scan the methods of that window object for the specified method, then the methods of the containing window's class. The same applies to report objects and menu lines.

Note that 'Do method' can not be used to invoke methods in other instances or classes. It is unique in that it can invoke both the methods of an instance (e.g. calls within a window instance) and the methods of a non-instance (i.e. calls within a code class).

When used in an instance, 'Do method' should only be used to invoke the PRIVATE methods of that instance (i.e. those without a leading '$'). It should not be used to call the instance's public methods because it does non-virtual dispatching (i.e. static binding), which defeats polymorphism. The following examples demonstrate the problems and serve to illustrate the differences between the 'Do' and 'Do method' commands:
A) You create a window class A with methods $m and $n. Method $m calls $n.
You create a window class B as a subclass of window A, and you override method $n.
You create an instance of B named MyB, and execute the command "Do $iwindows.MyB.$m()". This invokes the method $m as defined in class A.

Now, if $m calls $n using 'Do method', it will invoke $n as defined in class A, instead of the version in B. However, if $m calls $n using the 'Do' command, it will correctly invoke the version of $n in class B.

B) You create a window class C with method $x.
You create a window class D as a subclass of C. You override method $x, and add method $y which calls $x. The override of method $x in class D uses the 'Do inherited' command to call the version of $x in class C.
You create an instance of D named MyD, and execute the command "Do $iwindows.MyD.$y()". This invokes the method $y as defined in class D.
Now, if $y calls $x using 'Do method', the 'Do inherited' command in $x fails (i.e. it does nothing). However, if $y calls $x using 'Do', the 'Do inherited' command correctly invokes the version of $x in class C.

Do code method

Syntax: Do code method <classname>/<methodname>(parms...) returns [resultvar]
or: Do code method <classname>.<methodname>(parms...) returns [resultvar]

<classname>

The name of a class. This will usually be a code class, but you can specify any class type that has methods.

<methodname>

A method in the specified class.

[resultvar]

An optional variable. You can set a reference (as with the Set reference command) by specifying an Item reference-type variable as the resultvar.

'Do code method' is similar to the OMNIS Classic command 'Call procedure <formatname>.<procedurename>'. It's usually used to call methods in a code class, but it can (and should) also be used when calling static class methods in other class types. (Static class methods are methods of a class that you can call whether or not any instances of that class exist, e.g. the Set & Get routines that you need to provide for accessing the public class variables, if any, in a class. Static class methods in OMNIS are equivalent to static member functions in C++. Note that a static class method cannot access instance variables defined in the class, because it isn't associated with any particular instance.)

Note that 'Do code method' cannot be used to invoke methods of an instance. It is intended mainly for accessing the global methods, if any, in your application, e.g. GetExchangeRate(), GetUserName().

When you execute a 'Do code method' from within an instance, $cinst does not change while executing the called method, i.e. it still refers to the calling instance.

Do inherited

Syntax: Do inherited returns [resultvar]

The 'Do inherited' command allows you to reuse code in a base class method when you extend its behavior in a subclass. That is, if you override a method in a subclass and then execute a 'Do inherited' command within that method, it will invoke the similarly-named method in the base class. The 'Do inherited' command can be executed at any point in the subclass method, i.e. the beginning, the end, or somewhere in the middle. The base class method has normal access to any parameters passed to the subclass method. Note that you should not change the overriden method's signature (i.e. the types, number or order of its parameters) in your subclass since OMNIS does not support method overloading.

For example, consider the following method, as defined in a base class and its subclass:

```
Class:   SalariedEmployee
Method:   $computeWages(startDate, endDate)
{Get the number of hours worked}   ;; Get the employee's
hours for this period (using a query, etc)
Calculate wages as hours * hourlyRate       ;; Compute
salary
Quit method wages    ;; Return the result

Class:   SalariedEmployeePlusCommission   (subclass of
SalariedEmployee)
Method:   $computeWages(startDate, endDate)    ;;
Same name and parameter list as in the base class
Do inherited returns wages   ;; Compute base salary using
 the method in SalariedEmployee
{Get the employee's sales figure}    ;; Get the employee's
sales for this period (using a query, etc)
Calculate wages as wages + sales * commissionPercent
;; Now add the commission income
Quit method wages    ;; Return the result
```

If you execute a 'Do inherited' command in a method that isn't an override of a base-class method, it does nothing.

Do default

Syntax: Do default returns [resultvar]

The 'Do default' command allows you to invoke the 'built-in' version of a standard notational attribute or function that you've overriden. OMNIS Studio allows you to override the reading and/or writing of many standard attributes.

To override the reading of a standard attribute value, you need to provide a method with the same name as the attribute at the appropriate level, e.g. to customize the value returned by the $width attribute for a particular window class, you need to create a class method named '$width' in the window. For example:

```
Method:  $width
Do default  returns lWidth     ;; Call the standard version
of $width to get the width in pixels
Calculate lTempString as pick((lWidth > 200) + (lWidth >
400), 'Narrow', 'Medium', 'Wide' )
Quit method  lTempString     ;; Return the result
```

This custom version of $width returns a text value instead of the usual numeric value. To test it, add the following code to the window's $event method:

```
On evResized
        OK message {Window width = '[$cwind.$width]'}
;; Resizing the window will display the dialog
```

Similarly, to override the writing of $width in the same window class, you need to create a class method named '$width.$assign'. For example:

```
Method:  $width.$assign       (Parameter: pWidth)
Do default      ;; Call the standard $width.$assign to set
the window width
If ($cwind.$left+pWidth)>(sys(104)-18)     ;; Check if the
window's right edge is now off the screen
   Calculate $cwind.$left as max(0,sys(104)-18-pWidth)
;; If so, move its left edge accordingly
End If
Quit method kTrue     ;; Return kTrue to indicate that the
assignment worked
```

To use the custom version of $width.$assign, execute this code:

```
Do $iwindows.$add('MyWindClass', '*')  returns lRef
;; Create an instance of the window class
Do lRef.$width.$assign(400);; Change the window width
using the custom $assign
Calculate lRef.$width as 400 ;; Another way to do the
same thing
```

Note that 'Do default' can only be used in a method with the same name as a standard notational attribute or function. In addition, the method has to belong to the same application element (e.g. a window object, a report, etc) as the attribute. Lastly, the attribute or function has to be overridable. If all these requirements aren't met, the command does nothing.

Do redirect

* Syntax: Do redirect <instance> returns [resultvar]

<instance>:

A reference to an instance. It can be specified using:

1) notation, e.g. $cinst.$objs.DependentObject, $itasks.MyHandler, etc
2) an Item reference variable, or
3) an Object variable.

The 'Do redirect' command can only be executed from within an event handling method, i.e. $event or $control (or a method that is called by either). When executed in a $event method, it calls the $event method of the specified instance, passing it the current event. Similarly, if executed in a $control method, it invokes the $control method in the specified instance. After the specified event handler completes, execution continues in the calling routine. The command returns kTrue if the instance was found, otherwise it returns kFalse.

The standard OMNIS event-handling hierarchy is containment-based, e.g. a click on a window object is first reported to the $event method of that object, then (if passed on) to the containing window's $control method, then (if passed on) to the containing task's $control method. 'Do redirect' can be used in situations where (a) you need to pass events up some other, developer-defined chain, or (b) you need to change the event-handling chain dynamically. For example:

1) You could switch handler objects to implement different modes of behavior for a window.
2) In a spreadsheet-type window, you could pass events occurring in one cell to all of its dependent cells, i.e. the cells referring to that cell.

The Omnis Advisor

by Jim Pistrang

Welcome to the fourth installment of the Omnis Advisor. There have been some threads on the list lately concerning parameters and prompt windows in the Omnis Classic environment. In this column and demo I'll discuss using parameters to create generic prompt windows, which can be called from anywhere within an application and return dates, numbers, and text. At the end of the column, I will pose the task to be discussed in the next column. I encourage all readers to send me their solutions, comments, and questions at jim@jpcr.com.

The Task

How can I

a) correctly use parameters, field names, and item references?

b) create a simple prompt window 'object' that can be called from anywhere?

The Demo

To best understand this discussion, download the JPCRDEMO.LBR from the following URLs:

windows: ftp://ftp.crocker.com/pub/users/pistrang/JPCRDEMO.zip

mac: ftp://ftp.crocker.com/pub/users/pistrang/JPCRDEMO.sea.bin

The file that you download contains a read-me, this article (as well as the prior Advisor articles), and two libraries, named JPCRDEMO and NEWVER. To test the parameter functions, just open the library!

The Solution

Click on the Parameters button on the Main Window of JPCR Demo to go to the Parameters Demo Window.

In the simple example on the left, the user clicks on a button named 'Prompt for Date'. A prompt window is opened, where a date can be entered. When the prompt window is closed, the date is returned to the top window.

Here's what is happening under the hood: The Parameter Demo window includes a format variable named fvRDate. This is the field that the date will be returned into. When the user clicks on the 'Prompt for Date' button the following code is executed:

```
Call procedure wdPrompt2/init (fvRDate,'Please enter a date!','date:','Date Prompt Window') {init}
Redraw named fields fvRDate to fvRDate
```

In the call, four parameters are passed. The first is the field in the calling format that we want our data to be returned into. In this example the field is a format variable defined in the calling window, but any variable type will work (even a local variable). The other three parameters are text strings...the first is the message that will appear on the prompt window, the second is a field label that will appear on the prompt window, and the third is the window title of the prompt window.

The called procedure in the prompt window looks like this:

```
Other parameters are optional
Parameter pvPassedField (Field name)
Parameter pvMsg (Character  10000000)
Parameter pvLabel (Character  10000000)
Parameter pvTitle (Character  10000000)
Set reference fvReferenceField to pvPassedField.$ref
Calculate fvMsg as pvMsg
Calculate fvLabel as pvLabel
Clear format variables when closed
Open window [$cformat.$name]/CENTER
Set top window title {[pvTitle]}
Enter data
Close top window
```

The first receiving parameter, pvPassedField, is defined with a field type of 'Field Name'. This means that any action against pvPassedField will really be taken on the first parameter in the calling procedure, which in this case is fvRDate from the Parameter Demo Window. The following three parameters are simple text strings. pvMsg and pvLabel are moved into format variables so they can be displayed on the Prompt Window, and pvTitle is used in a Set Top Window Title command after the window is opened.

Note what happens with the pvPassedField parameter. Elsewhere in the Prompt Window a format variable named fvReferenceField has been declared, with a field type of Item Reference. The Set Reference command in the above procedure sets the reference for fvReferenceField to pvPassedField. This means that whenever fvReferenceField is modified on the Prompt Window, the field that fvReferenceField references is actually being modified. This field, pvPassedField, in turn references the field passed in the call to the Prompt Window. Therefore, when the user enters data into the prompt field on the Prompt Window, they are actually modifying the field in the calling window.

Following this model, the other buttons in the example prompt for a number, a character string, and a yes/no response, all calling the same Prompt Window, but passing different parameters.

The demo includes a more complex example as well, in which the Prompt Window contains multiple fields, which are hidden or visible based on a message passed from the calling procedure. Check it out!

Next Column

My next column topic is unknown at this time. I'll keep my eye on the Omnis List, and try to come up with something that comes up on a regular basis. If you've got a suggestion, send me email!

Jim Pistrang is the president of JP Computer Resources, based in Amherst, Massachusetts. He's a full time Omnis developer, a Certified Omnis Software Professional, an Omnis Ambassador, and the co-director of the Boston Omnis User Group. Contact him via email at jim@jpcr.com, or visit him on the web at http://www.crocker.com/~pistrang/.

OMNIS

advanced it could only
(Or did it?)

Get ready for a close encounter of the eighth kind. Mac® OS 8 is here. And with it comes a whole new way of working with your Mac OS computer.

Mac OS 8's purpose on this planet is to help you work more productively, and remove the barriers between you and your full potential. And it does this with performance that's out of this world. Thanks to the Mac OS 8 improved multi-tasking Finder™ software, you can accomplish several tasks at once, like opening and copying files without any waiting around. But all this power is nothing without organisation. That's why its new unparalleled customisation features let you adapt your system to match your individual work style. Spring-open folders, contextual menus and pop-up windows help get you around faster, and are part of a new 3-dimensional look that's even easier to work with. And Mac OS 8 has integrated Internet capabilities. Internet Explorer, Netscape Navigator,™ the PointCast Network and a new Internet Setup Assistant are built right in, making it easier than ever to get on the Net, to email, surf or browse the cosmos.* All this lets you work the way you think, so you can be more efficient. It even includes QuickTime® multimedia technology. And to show that Mac OS 8 comes in peace, it can read Windows files, whether you have the original application or not. It's also compatible with your existing Mac applications. Mac OS 8 is designed for 68040 and PowerPC™-based systems, and features reliability improvements that'll have you glad you went with the power of 8.

Intelligent life is out there. Mac OS 8 is proof.

For your own close encounter with Mac OS 8, visit our Web site at http://www.apple.com.au for more information and the names and addresses of your local resellers. Or give us a ring at 1-800-025 355.

Mac OS

AHEAD OF THE CURVE

What a c ... akes.

OMNIS
software
inc.