

Personal Systems

IBM'S MAGAZINE FOR PC PROFESSIONALS

MARCH/APRIL 1996



SOM Technology: Making the Pieces Fit

SOM's Language Neutrality

Distributed SOM

SOM and OpenDoc

BULK RATE
U.S. POSTAGE
PAID
PERMIT 1016
FORT WORTH, TEXAS

IBM



FILESTAR/2™

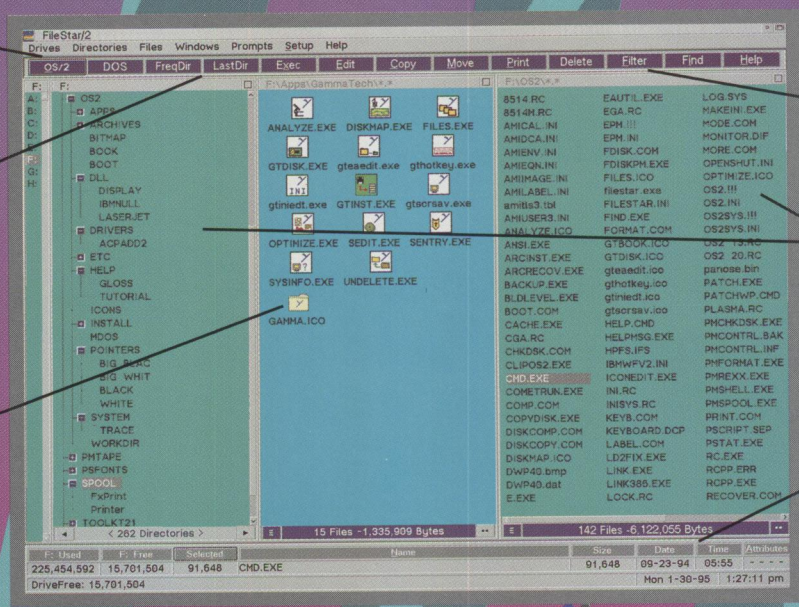
File Manager for OS/2

Exploring the vast galaxy of information requires tools to navigate disk space. FILESTAR/2 sets you in the cockpit, allowing you to chart your file system.

Operations can be performed from pull down menus, the button bar, hot keys or convenient pop-up menus.

Previously used directories are stored for quick access, or create your own customized frequent directories list.

Files can be displayed by name, icon, name and icon, or in details view.



Set up filters to display only the files you choose.

Drag and drop files between file windows, or drop files onto desktop objects.

Double-duty labels function as push buttons: click on the Attributes, Date or Time labels to pop-up a dialog box to modify the file timestamp or attributes.

- Query directories for the number of files and total space allocated.
- Continuously monitor the size of the swap file, amount of available memory, and free space.
- Perform file operations effortlessly.
- Compression utilities allow you to quickly compress/decompress files and directories and browse the contents.

(800) 944-3028



(405) 632-6537 FAX

SofTouch Systems, Inc., Workstation Division

1300 South Meridian, Suite 600, Oklahoma City, Oklahoma 73108

Specialists Since 1989

Circle #15 on reader service card.

Lock and Load!

CENTRALLY MANAGING, SECURING, AND DELIVERING AN OS/2 DESKTOP OVER ANY LAN BY ERIC OSMANN

OS/2 has become a corporate workhorse since its inception several years ago. Yet many questions have arisen on how to manage the workplace shell and, most important, secure it and the applications it helps launch.

There are a few tools which advocate their ability to manage the shell - most work best in a stand-alone mode. Still, the corporate environment demands efficient centralized management and security optimized for networks. This requirement, combined with my background in development of the OS/2 workplace shell, has led to a careful examination of OS/2 security and commitment to The Desktop Observatory by Pinnacle Technology. This paper discusses the requirements and the features built into Pinnacle's unique technology.

Requirements

Although pure file level security requires IBM's Security Enabling Services (Pinnacle has been a member of Beta efforts and is architected to support these key features already available for 2.11), there are critical elements in OS/2 security that go well beyond the simple ability to restrict right mouse button options. These include: 1) ease of creating desktops; 2) network design; 3) additional security elements; and 4) flexibility.

Ease

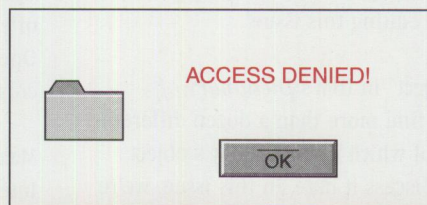
Pinnacle has developed a powerful SOM based tool called "Finder". Finder allows an administrator to quickly create a desktop, snap a picture of it (including all of those oblique DOS settings) and save it to Pinnacle's bandwidth-sensitive file (patent pending). Files can, of course, be saved anywhere on the network and instantly be built at any machine. There is no need for an administrator to know REXX, although those who do will appreciate Pinnacle's ability to associate

Eric Osmann was a member of OS/2 Workplace Shell development team at IBM and left in late 1995 to lead development at Pinnacle Technology, Inc. He resides with his family in Colorado Springs and can be reached through electronic mail at helpdesk@pinnacletech.com.

"events" (programs, etc.) with objects and the pure speed at which it builds desktops.

Network Design

The Desktop Observatory was designed with networks in mind. Administrators will appreciate Pinnacle's small file size of less than 10k per desktop and its ability to work over any protocol and NOS. With this bandwidth, lap tops can be secured over a phone line!



The Desktop Observatory is able to limit users to a centrally managed specific list of applications. User attempts to cleverly create objects via ProgMan, word processors, or REXX will result in a security violation!

Another feature of the Desktop Observatory is that it allows administration of OS/2 clients with the current security server for network based applications. Regardless what NOS is being utilized by a firm, The Desktop Observatory will not build an object unless it can resolve its security checks. It cannot resolve these check unless users are correctly authenticated. The result is a user gets his/her appropriate desktop *whenever and wherever* they log on and the administrator does not have to redundantly maintain rights. No desktop need be provided until after a user has been authenticated (those interested in Single Sign-On should contact Pinnacle).

More on Security

Auditing is a common requirement for the security conscious. A full capability is built into The Observatory. This is integrated with the security daemon which includes the ability to restrict clever users from building their own objects to violate client security. Also, great pains are taken to reduce all exposures between power-on and the workplace shell. A special program was created to start

the shell (dskshell.cmd), insure networking starts correctly, and inhibit user's ability to interrupt the process via a cntl-break. Disabling the Alt-Fx function is a feature that can be engaged, if desired. Users can enable their hardware boot protection in the bios or call Pinnacle for a centrally distributable software solution.

Flexibility

There are circumstances where corporations do not need high security and only need the ability for users to get the appropriate desktop. In fact, this may be combined with the need for users to be able to modify their own desktop yet move from machine to machine—or what some customers refer to as a "follow me" desktop. Pinnacle uses MID (Mobile Independent Desktops) to provide this flexibility. MID simply utilizes the Finder function mentioned earlier to "snap" the desktop at log off and save the appropriate file to the network so it can be delivered at the next log on. This process applies to PowerPC desktops as well (the Observatory is provided on the PowerPC-OS/2 Application Sampler). End Users should not care what type of OS/2 machine they log onto and with Pinnacle's technology they don't have to.

Summary

Whether you are interested in security or centralized management and delivery of OS/2 desktops, the technology available through Pinnacle can save a great deal of heartache and headaches.

Pinnacle Technology, Inc. was founded in 1992 to bring four key elements to OS/2: 1) Centralized Management; 2) Security; 3) user-proof; and 4) bandwidth sensitivity. Pinnacle's unique means of efficiently delivering desktops (patent pending) is used on over 100,000 OS/2 workstations worldwide. Current products include The Desktop Observatory V4 (centralized desktop management and security), The Desktop Commander (centralized desktop management), and Kid Proof/2 (for stand-alone machines). They can be reached at (800)-525-1650, (317)-279-5157, www.pinnacletech.com, or ehennig@pinnacletech.com.

The above is an advertisement. Copyright 1995, Pinnacle Technology Inc.

See us at Booth 515 at the IBM Technical Interchange Conference
Circle #17 on reader service card

The Object of This Issue



Back when I was in school (no unkind comments, please), an *object*, as drilled into me by my mother and my aunt (both of whom, as teachers in the school I attended, suffered to teach me English grammar) was a noun that was the result of the action of a verb or a noun equivalent in a prepositional phrase. As an editor, that definition is still the first one that comes to mind when discussing objects. So you can imagine the paradigm shift I had to make when editing this issue!

If you look up "object" in IBM's *Dictionary of Computing*, you'll find more than a dozen different definitions—none of which defines today's object technology as we discuss it here. In this issue, we're talking about putting objects or "parts" together to make a whole—a complete application. If you're into developing applications, you're going to want to read every word. Even if you're not a hard-core developer, you'll learn how object technology is making your computing activities easier and less expensive.

This issue's authors teamed up to ensure they covered a wide range of SOM topics. They all used the same sample program—a valet parking system—making it easier for you to understand the similarities and differences in each of the articles. In some cases, the code samples were far too lengthy to include in the magazine. However, you'll find all the code samples on the World-Wide Web at <http://pscc.dfw.ibm.com/psmag/>.

IBM Technical Interchange '96

Even though you'll find a wealth of information about object technology and IBM's System Object Model in this issue, we've just shown you the tip of the iceberg. Where can you learn more? IBM's Technical Interchange '96 (April 22-26 at the Opryland Hotel in Nashville, Tennessee) provides not only many one-hour elective sessions on developing with objects, but this year the conference is offering five-hour tutorials, including sessions on OpenDoc, SOM, open distributed computing, and cross-platform application portability.

Make your plans now to attend the Technical Interchange, where you'll have the chance to meet and exchange ideas with your peers worldwide. There's over 300 elective technical sessions covering four platforms—AIX, AS/400, OS/2, and S/390—plus four keynote speakers, an exhibit hall filled with the latest applications, and, of course, some toe-tapping good times to be had in the capital of country music. For more information, call (800) 872-7109 in the U.S. or Canada, or (617) 893-2056 outside of North America.

Betty Hawkins, Editor

Look for the reader service card (more information for you) and the opinion card (so we'll know what you want) between pages 56 and 57. Fill them out and mail them today!

New Multimedia Sound Card for Micro Channel

New
Product

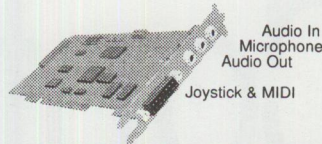
Innovative 'ChipChat Sound Card' provides state-of-the-art multimedia audio for Micro Channel computers. Supports DOS, Windows, OS/2, and AIX . . .

INTRODUCING THE ChipChat® Sound Card, an exciting new product that provides state-of-the-art multimedia audio for Micro Channel computers.

Add this sound card to your PC and hear CD quality audio from your favorite multimedia programs.

The ChipChat Sound Card fits into a full or half sized slot on a Micro Channel® computer. It's hardware compatible with just about every DOS game and educational software out there, including those that require SoundBlaster® compatibility.

The ChipChat Sound Card comes with it's own high performance software drivers with 16-bit audio for Windows® and for OS/2®. It also "works like a charm" with the 8-bit audio SoundBlaster drivers which are shipped as standard with OS/2.



ChipChat Sound Card for Micro Channel

State of the Art Music Synthesis

The ChipChat Sound Card provides state-of-the-art music synthesis in two forms: FM and WaveTable.

FM uses mathematical formulas to emulate the sound of musical instruments. FM synthesis provides good quality sounds and is economical.

WaveTable synthesis stores 128 actual musical instrument samples on a tiny chip, so it makes music that sounds great - just like the actual instrument! WaveTable is truly state-of-the-art, and is the "method of choice" of the music industry.

Exceptional Product

The ChipChat Sound Card comes in two different models: The ChipChat Sound-16 with FM for \$199, and the ChipChat Sound-32 with FM and WaveTable for \$259. If you buy the Sound-16 and later decide you want WaveTable, an upgrade is available.

The ChipChat Sound Card is designed and manufactured in the USA and has been subjected to rigorous tests to guarantee a solid and exceptional product.

The ChipChat Sound Card is available direct from ChipChat Technology Group (313-565-4000), and comes with the protection of the 30-day moneyback ChipChat Guarantee.

Ordering information is at the bottom of this page. Further information is at <http://www.ChipChat.com>

Send messages to wireless pagers from OS/2

Practical App
for OS/2

Combine the Power of OS/2 with the Freedom of Wireless Messages . . .

THE CHIPCHAT WIRELESS COMMUNICATOR is an exciting software product that sends text messages to pagers directly from OS/2.

ChipChat works with pagers from any paging company, including Air-touch, Ameritech, MobileComm, PageNet, SkyTel and many others.

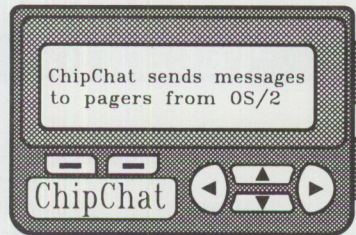
Remarkably easy to use

ChipChat has an easy-to-use, workplace shell interface. You simply drag a ChipChat Pager object out of a template, configure the settings, and start sending messages!

Page-Enable your applications

ChipChat can also send pager messages from other applications, including Rexx, and the command line!

ChipChat is ideal for LAN Administrators who want to receive an informative text page when their network needs attention!



Versatile, Easy, Reliable.

ChipChat is advanced multi-threaded 32-bit object software based on IBM's SOM technology. It's been "through the wringer" with extensive corporate beta testing and has passed a suite of rigorous tests set by IBM. ChipChat is certified as "Ready for OS/2 Warp - NSTL Tested".

Join thousands of customers in the USA and abroad who are successfully using ChipChat for their OS/2 wireless paging needs!

How much? Just \$79

ChipChat Wireless Communicator is available direct from ChipChat Technology Group for only \$79 and comes with the protection of the 30 day moneyback ChipChat Guarantee.

Ordering information is at the bottom of this page.

Complete information is at the ChipChat Internet Web site: <http://www.ChipChat.com>.

The ChipChat Guarantee

If you're dissatisfied with a ChipChat product for any reason, if it isn't everything we say it is and more, then return it within 30 days for a prompt, cheerful refund.

Phone **313-565-4000**, Fax **313-565-4001**, or Web <http://www.ChipChat.com>

How to Order
a ChipChat

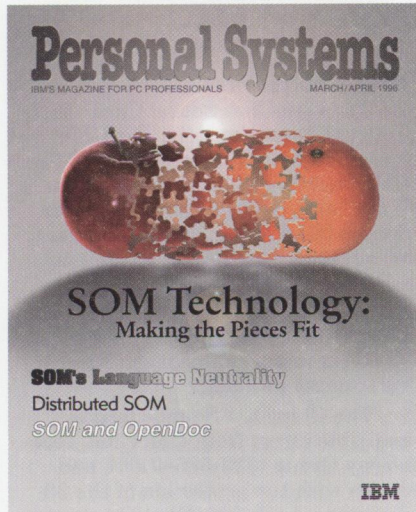
- ChipChat Sound-16 Card** with advanced FM music; DOS, Windows, and OS/2 support (AIX drivers are optional); CD quality 16-bit sound; 12-channel audio mixer; MPU-401 MIDI; Dual joystick port. (Can be upgraded to include WaveTable). **\$199.**
- ChipChat Sound-32 Card** - all the features of the ChipChat Sound-16 *PLUS* Incredible WaveTable music. **\$259.**
- ChipChat Wireless Communicator** software with the easy-to-use workplace shell interface, powerful command line interface, ability to 'Page-Enable' your applications, and ability to 'Page Enable' your Internet WWWeb site. **\$79.**

We accept Visa, MasterCard, and American Express. Add \$10. for shipping.

© Copyright 1995 **ChipChat Technology Group, 24224 Michigan Avenue, Dearborn, Michigan 48124 USA**
ChipChat-Japan, Koga, Fukuoka Japan, phone: 81-(092)-943-0798 fax: 81-(092)-944-2253



Circle #18 on reader service card.



ABOUT THE COVER

Noted Dallas artist Bill Carr captures the essence of SOM's flexibility and power in this issue's cover. SOM transforms your application development efforts, letting you build robust, reusable, distributed applications.

Personal Systems Advertising Representatives

Personal Systems accepts paid advertising for applications, products, or services that run on or complement IBM's personal computer hardware and software products. To obtain a media kit and advertising rate information, contact one of the Personal Systems advertising sales representatives at the address below.

Lewis Edge & Associates, Inc.
366 Wall Street
Princeton, NJ 08540-1517
(800) ADS-4PSM

Winfield BoyerExt. 124
Lewis Edge, Jr.Ext. 123
Peter GriffinExt. 126
Joseph TomaszewskiExt. 125

Fax (609) 497-0412
CompuServe 72457,3535
Internet l_edge@netins.net
MCI Mail 275-0987
Voice (609) 683-7900



Printed on partially recycled paper using soy-based inks and may be recycled.
Printed in U.S.A.

Contents

FOCUS

6

What's New?

This issue's featured products include security software, data management products, hardware and software designed for those with special needs, tax software, and Internet-related products and information. It also highlights several books with topics ranging from LAN management to answers for OS/2 Warp FAQs.

18

Seton Hall Students Lead the Way From the Wireless to the Real World

"The classroom of the future" is reality right now at Seton Hall University in South Orange, New Jersey. Patrick Karle details the partnership between Seton Hall and IBM's Microelectronics Division, resulting in a state-of-the-art infrared wireless classroom for a group of high-potential students.

TECHNICAL

27

IBM System Object Model—The Wave of the Future (and Now!)

How does IBM's System Object Model (SOM) fit in the object world? What does SOM mean to you as a developer? This article answers these questions, plus defines the sample valet parking application used by the other articles in this issue.

30

Building SOM Objects with Native C++

Although using SOM with interface definition language (IDL) is great for new applications, what about existing C++ applications? Will you need to rewrite your class definitions from C++ to IDL to take advantage of SOM and DSOM? With Direct-to-SOM (DTS), the answer is a reverberating NO! See how you can use Direct-to-SOM to build SOM objects directly from C++.

34

Distributing Objects with DSOM

This article provides you with background information for distributing objects with Distributed SOM (DSOM) and walks you through the necessary changes to convert a sample application into a DSOM application.

40

Using OpenDoc and SOM in Application Development

The initial version of OpenDoc is now available for OS/2. Learn about OpenDoc and review the content of a simple "Hello World" component.

LITTLE SOLUTIONS

69

Corrective Service Information

Refer to this section for the latest maintenance release levels and other software service information.

IBM Personal Systems Technical Solutions is published bimonthly by Personal Systems Competency Center, International Business Machines Corporation, Roanoke, Texas, U.S.A. Send any correspondence and address changes to *Personal Systems* at:
IBM Corp. Mail Stop 01-04-60
5 West Kirkwood Blvd.
Roanoke, TX 76299-0015

Personal Systems can be found on the Internet's World-Wide Web at: <http://pscc.dfw.ibm.com/psmag/>

IBM customers are eligible to receive the magazine free of charge and can request subscription information by mail or Internet, or by faxing a request to (817) 962-7218.

© Copyright 1996 International Business Machines Corporation

21

**"Out, Damn Spot!"
or How to Rid Your OS/2 Desktop of Pesky Programs**

Having trouble getting rid of old applications? Want to clean up your INI files? Need to split your EAs? Well, before you resort to Lady Macbeth's removal methods, find out how SofTouch Systems' UniMaint will take care of these pesky problems for you.

23

Why SOM?

This article defines and overviews IBM's System Object Model (SOM) to give you a better understanding of its impact on you, whether or not you're an application developer. After defining SOM in detail, the article highlights its technical strengths and significance in the marketplace.

50

Enabling Industrial-Strength OO Applications with SOM and CORBA services

This article discusses the central role that IBM's System Object Model (SOM) and Object Management Group's (OMG's) CORBA services specifications play in developing robust, reusable, distributed applications. See what you can do today to prepare for tomorrow's distributed development environments.

58

SOM Language Neutrality: A VisualAge for Smalltalk Perspective

This article addresses how to use VisualAge for Smalltalk (VAST), a premier workstation application development tool, to provide a graphical user interface front end to a sample application.

63

SOM Language Neutrality: An OO COBOL Perspective

By describing an object-oriented COBOL program using SOM-enabled classes written in C++, this article illustrates the language neutrality of SOM objects and introduces many of COBOL's new object-oriented language extensions.

Editor and Publisher

Betty Hawkins
(817) 962-5799
bhawkins@vnet.ibm.com

Assistant Editor

Lia Wilson
(817) 962-6267
lia@vnet.ibm.com

**Business Manager/
Circulation Manager**

Van Landrum
(817) 962-5810
vlandrum@vnet.ibm.com

Editorial Assistant

Jeffrey Miller
(817) 962-5823
jeffreym@dalvm41b.vnet.ibm.com

Subscription Manager

Rose McAlister
(817) 962-5821
rmcalister@vnet.ibm.com

Business Processes Assistants

Dustin Lyon
Terri Sharp-Kubica

**Publication Services,
Typesetting, and Design**

Terry Pinkston/Corporate Graphics
Arlington, Texas

Illustrator

Bill Carr
Dallas, Texas

Printing

Dave Willburn/Motheral Printing
Fort Worth, Texas

Editorial Services

Mike Engelberg/Studio East
Boca Raton, Florida

Executive Publishers

Beau Sinclair and Beverly Montgomery

Copying or reprinting material from this magazine is strictly prohibited without the express written permission of the editor. Titles and abstracts, but no other portions, of information contained in this publication may be copied and distributed by computer-based and other information service systems.

What's New?

Tape Backup Solutions for OS/2



Conner Tape Products Group, a division of Conner Peripherals Inc., offers three high performance tape backup solution packages for OS/2 Warp.

Standing one inch high, these 3.5-inch minicartridge systems offer 4 GB of compressed capacity with a data transfer rate of 27 MB per minute. Supporting both SCSI and IDE interfaces in the OS/2 market, Conner's internal **CTM3200 SCSI**, **CTM4000 IDE**, and external **CTM3200E** drives are available with the comprehensive, easy-to-use features of Computer Data Strategies' **Back Again/2** backup and recovery software.

Back Again/2, designed exclusively for systems running OS/2, features an intuitive graphical interface, a built-in scheduler, a stand-alone restore facility, password protection, and "drag and drop" file selection flexibility. The new Conner tape systems back up and restore HPFS and FAT files and EAs (extended attributes), plus LAN Server and NetWare file servers.

Designed to use Sony QIC-Wide media, Conner's high performance minicartridge tape drives provide reliable, high capacity data storage on a single, low cost cartridge, making these drives a cost effective backup solution.

For more information, circle 1 on the reader service card.

IBM's Independence Series of Products



IBM has developed a number of helpful devices and software tools that make the computer more accessible and friendly to people with vision, hearing, speech, mobility, and

attention/memory impairments. These products include interactive speech and cognitive therapy tools, screen access software, keyboard access utilities, and screen enlargement programs.

AccessDOS

This powerful set of utility programs is designed to give people with mobility, vision, and hearing impairments greater access to the computer. By providing extended keyboard, mouse, and sound access, AccessDOS enables IBM DOS users to operate a computer with more ease and efficiency, thereby increasing employability.

AccessDOS is especially helpful to anyone with special needs, including eye/hand coordination difficulties or mobility impairments. For many people, AccessDOS makes using a keyboard possible for the first time.

Features include:

- **StickyKeys** solves the problem of having to press two or more keys simultaneously. One key at a time does the job.
- **MouseKeys** makes it possible to use the numeric keypad to move the cursor around the screen, thus simulating the use of a mouse.
- **RepeatKeys** allows you to set how fast keys repeat when held down.
- **SlowKeys** enables you to set the sensitivity of the keyboard. This feature instructs the computer not to accept a key as "pressed" until it has been held down for a specified length of time.
- **BounceKeys** prevents double characters from being typed if you bounce on the key when releasing it.
- **SerialKeys** allows you to control the keyboard and mouse using optionally available input devices.
- **ShowSounds** (for the hearing impaired) causes the screen to blink or display a small musical note when the computer makes a sound.

- **ToggleKeys** (for the vision impaired) provides a sound to indicate when the Caps Lock, Num Lock, or Scroll Lock keys are activated.

KeyGuard

KeyGuard makes it easier for people who have mobility impairments to use the keyboard. This plastic overlay snaps onto several types of IBM keyboards, creating depressions into which the finger or a typing stick can be easily guided. KeyGuard also acts as a hand support for users with motor coordination conditions.

Screen Magnifier/2

Screen Magnifier/2 displays screen print in an easy to read size for those with vision impairments. It enlarges text and images up to 32 times their normal size in OS/2, DOS, and Windows applications.

A feature called Focus Tracking makes the mouse pointer automatically move to the area of interest on a screen, saving time and enhancing productivity. To sharpen the clarity of text and images, Screen Magnifier/2 can reverse colors.

Screen Reader

Screen Reader converts screen information into speech, letting visually impaired individuals operate a computer with greater ease and efficiency. Screen Reader can benefit others as well, including people with a reading difficulty or those who prefer listening more than reading.

The two versions of Screen Reader (DOS and OS/2) work with a wide range of applications, including word processors and spreadsheets.

SpeechViewer II for DOS

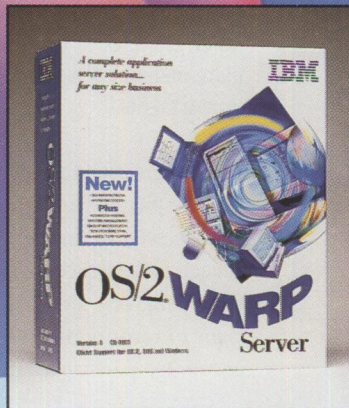
SpeechViewer II 1.2 is a versatile clinical tool for speech pathologists, teachers, and other professionals treating communication disorders. SpeechViewer II provides a multi-function speech therapy solution by integrating speech analysis features, motivational clinical exercises for direct client

First there was OS/2® Warp...

Then OS/2 Warp Connect...

NOW AVAILABLE: OS/2 WARP SERVER

The Complete Business Server Solution from IBM®



OS/2 Warp Server provides an integrated platform for the emerging application server environment as well as a complete set of traditional file and print services. OS/2 Warp Server combines the market-proven quality of OS/2 Warp and LAN Server 4.0 with a wealth of functional enhancements in systems management, backup, remote access and advanced printing options.

A POWERFUL SERVER

OS/2 Warp Server inherits from LAN Server 4.0 a sophisticated set of network capabilities, including an easy-to-use drag-and-drop administration model and customizable security features. Also included is a NetWare migration utility.

SOPHISTICATED SYSTEMS MANAGEMENT MADE EASY

OS/2 Warp Server will contain systems management features that provide remote support, software and hardware discovery, alert management and software distribution.

REMOTE ACCESS

A full set of remote access capabilities is included with OS/2 Warp Server. There's a wide range of connectivity options and multiple levels of security.

ADVANCED PRINT FUNCTIONALITY

OS/2 Warp Server includes new printing enhancements that will even utilize host printers.

CAREFREE SYSTEM BACKUP AND RECOVERY

IBM has implemented a comprehensive backup and recovery system in OS/2 Warp Server. An advanced disaster recovery feature will allow a business to recover vital data, even in the event of a complete server hard disk crash.

BROAD CLIENT SUPPORT

OS/2 Warp Server will support all prevalent network clients, including OS/2 Warp and OS/2 Warp Connect, DOS, Windows® 3.x, Windows 95, Windows NT, LAN Server and Macintosh.

OSIBM1637	OS/2 Warp Server Version 4	\$518	UGOSIBM1560	OS/2 Warp Server Version 4 Upgrade (from Lan Server 3.0 and OS/2 2.1)	\$328
OSIBM1638	OS/2 Warp Server Version 4, First Step (OS/2 Warp Server plus 10 users)	\$682	UGOSIBM1562	OS/2 Warp Server Version 4 Upgrade (from Lan Server 4.0 and OS/2 2.1)	\$254
OSIBM1639	OS/2 Warp Server Advanced Version 4	\$1069	UGOSIBM1565	OS/2 Warp Server Version 4 Competitive Upgrade	\$411

TO
ORDER
CALL:

Softmart®
The smart resource for business

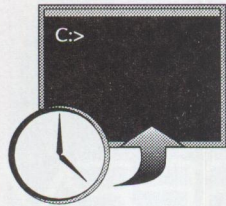


800-328-1319

OS/2 and IBM are registered trademarks of International Business Machines Corporation. All other product or company names are trademarks or registered trademarks of their respected companies. Softmart is a registered service mark of Softmart, Inc. COPYRIGHT © 1996 by Softmart, Inc.

Circle #19 on reader service card.

Schedule Programs & Reminders Automatically...



Chron v4.0 Scheduler for OS/2

Only \$99

Site Licensing Available

Uses:

- Schedule backups and database maintenance
- Remind yourself of recurring meetings
- Schedule long-running or resource consumptive tasks for after hours



Hilbert Computing
1022 N. Cooper
Olathe, KS 66061

Voice: (913) 780-5051
BBS/Fax: (913) 829-2450
CIS: 73457,365



Circle #22 on reader service card.

therapy, and clinical management features in an easy-to-use package.

With the clinical exercises, graphics, and microphone selection, clinicians can work with clients of all ages and with a variety of speech disorders. Clinicians can also use its clinical management features to track clients' progress and performance.

THINKable/DOS

For use in schools, hospitals, and rehabilitation facilities, this unique, interactive tool is designed to help individuals improve attention, memory, and discrimination by making learning fun and stimulating. This easy-to-use, innovative product uses an extensive variety of realistic computer graphics to focus attention and stimulate memory for people with cognitive impairments.

The user is asked to respond to colorful life-like pictures and graphics through touch-screen technology. It uses real speech for instruction, cueing, and reinforcement. The technology allows clinicians to record unique sounds or voices to encourage participation.

Using THINKable, processes are practiced in four critical areas that relate to daily living: visual attention, visual discrimination, visual memory, and visual sequential memory. To challenge the student's capabilities, the clinician can select the level of difficulty, stimulus duration, sequence length of various on-screen exercises, and reinforcement variables.

For more information, contact the IBM Independence Series Information Center at (800) 426-4832 (voice) or (800) 426-4833 (TDD) or view information on the World-Wide Web at www.austin.ibm.com/pspinfo/snshome.html.

Velcro Wire Harnesses



Polygon Wire Management Systems of Port Coquitlam, B.C. manufactures a line of

Velcro wire management products for organizing and securing the disarray of cables on telecommunication racks, workstations, computer networks, and cabling systems.

©1996 Cirrus Technology Inc.
Unite CD•Maker™

CD-ROM Mastering for OS/2 Warp

Unite CD•Maker™ lets you create your own custom CDs right from your desktop. Unite CD•Maker 2.0 includes these great features:

- Audio CD reading and writing
- Multi-session capability
- 32-bit APIs
- Long file name support
- Device drivers for many popular CD-recordables
- Beta OpenDoc™ interface upon request

Unite CD•Maker combines a multi-threaded design and drag-and-drop features for a powerful and easy-to-use CD mastering tool.

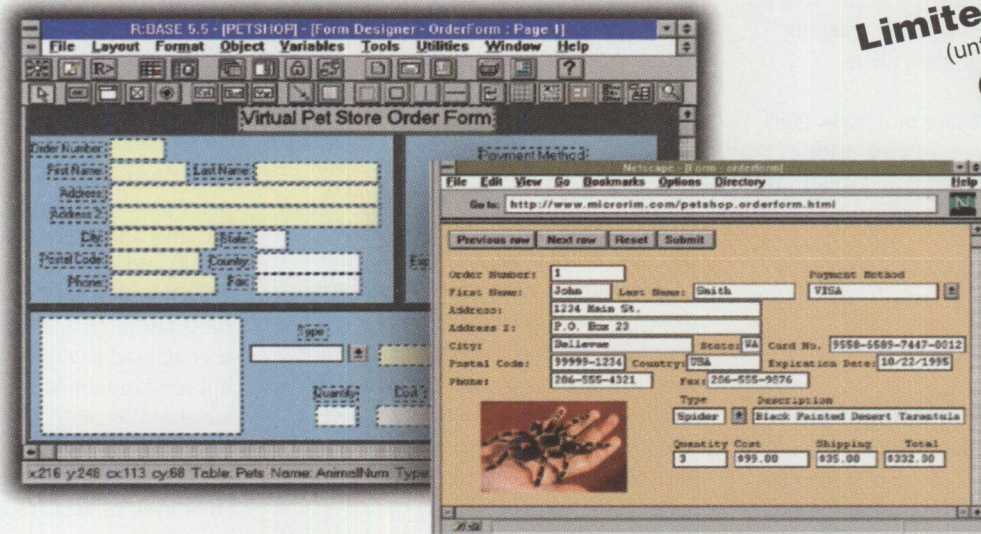
Cirrus Technology
Fourth Floor, 5301 Bucklestown Pike, Frederick, Maryland 21704
email: 102404.3643@compuserve.com internet: <http://www.cirrus.com>

1-301-698-1900

Circle #5 on reader service card.

It's so easy we could have called it "Your:WEB"

Limited Time Offer
(until April 15, 1996)
\$99



R:WEB - the first easy-to-use, affordable Internet database solution.

CREATE DOCUMENT

Making a database accessible to visitors of your Web site used to take a good understanding of scripting and a fair chunk of change. Not any longer. Now you can use R:WEB to effortlessly create forms for data exchange between your Web site and your database. R:WEB writes the code - you don't need to know HTML, CGI or PERL scripting. And your database can reside in any ODBC-compliant data source for R:WEB to dynamically link to it. R:WEB maintains security over your sensitive information through data integrity rules, constraints and password security - you have complete control over the information a Web browser can or cannot access. Best of all, R:WEB is built on the powerful R:BASE engine, one of the most respected database engines available. And, oh yeah, did we forget to mention? It's affordable.

So make your Web site work harder the easy way. With R:WEB. And check out the FREE CD-ROM offer.

<http://www.microrim.com>

MICRORIM®

A Subsidiary of Abacus Software Group
15395 SE 30th Place, Bellevue, WA 98007
(206) 649-9500

R:WEB available through your local reseller or distributor.

Copyright © 1995 Microrim, Inc., a subsidiary of Abacus Software Group. All rights reserved. R:BASE, R:WEB and Microrim are trademarks and registered trademarks of Microrim. Microrim reserves the right to change product and services offered at any time without prior notice.

FREE CD-ROM
If you're responsible for Web site management, we'll send you a FREE CD-ROM with sample R:WEB software.
Just call 1-800-628-6990, ext. 130



Circle #20 on reader service card.

By safely securing wires, particularly category five and fiber optic cables, you can prevent costly damage to them from kinks caused when they are bent through tray sections, relay racks, and storage cabinets or are pulled from their connectors due to excessive strain. Damaged wires can distort and cancel data transmission.

Polygon's Velcro systems are designed for instant installation using self-adhesive or screw mounts. Unlike nylon tie wraps and rigid systems, these proven secure systems are sensitive enough to avoid crushing category five wire or glass fibers.

Polygon offers a number of devices that are adaptable to most network setups. Polygon also offers customized solutions for more challenging situations.

For more information, circle 2 on the reader service card.

Fax Boards Support



Brooktrout Technology, a fax and voice messaging support provider, has announced that

its **TR114 Series** multi-channel fax boards are now supported by the **Lotus Fax Server**. You can use these multi-channel fax boards with the Lotus Fax Server for **cc:Mail** and **Notes** fax services. Lotus cc:Mail is the world's best selling LAN-based messaging system, and Lotus Notes is the industry standard client/server platform for developing and deploying groupware applications.

Brooktrout's TR114 Series boards offer advanced fax processing on a multi-channel board with two to eight ports per board. TR114 boards are available in a wide variety of analog and digital configurations. They offer advanced fax functions, including 14.4 Kbps transmission and reception, MMR, MR, and MH compression, error correction mode operation, and support for T.434 binary file transfer. Ideal for enterprisewide faxing, these fax boards feature integrated on-board DID routing as well as support for other routing techniques, including DTMF and T.30 sub-address routing. The TR114 also provides full voice processing support, including speech record and playback, and DTMF detection.

For more information, circle 3 on the reader service card.

Color Network Laser Printer



The **Optra C** from Lexmark brings you high-quality color printing, networking support, and network print management capabilities in one printer. Your workgroup can do everything they always did in black and white—letters, charts, graphics, presentations, and reports—with the full impact of color.

Its photographic-quality color printing on plain paper and transparencies eliminates the time and expense of outsourcing color proofs and multiple high-quality copies. The Optra C prints color in 600 x 600 dpi (dots per inch) and continuous tone, with user-adjustable gloss levels. In 600 x 600 dpi resolution mode, Lexmark applies advanced screening techniques to deliver results you'd expect in higher-priced color printers. ColorGrade ensures vibrant color even in areas of fine detail, and

IF YOU KNOW OS/2® YOU SHOULD KNOW INDELIBLE BLUE

SOMobjects Development Toolkit

by IBM

Give your applications a greater degree of openness than ever before. IBM SOM objects gives professional programmers a comprehensive toolkit for creating OO class libraries and instances of those classes using SOM and DSOM technology. Objects can be used and reused independent of the programming language used to create them.

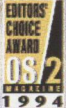

10H9767 SOMobjects DTK v2.1 Wt: 1.13
MSRP \$365.00 **\$255.00**

10H9769 SOMobjects Wkgrp Enab v2.1 Wt: 2.0
MSRP \$235.00 **\$179.00**

10H9780 DTK v2.1 UG from 2.0 Wt: 5.0
MSRP \$50.00 **\$39.00**

The DeskMan / 2™ Productivity Pack

from **\$74** Development Technologies, Inc.

DevTech, maker of the award-winning DeskMan/2, has created the powerful DeskMan/2 Productivity Pack. Combined with DeskMan/2 v. 1.51 (and a free upgrade to v2.0, when available), an outstanding selection of invaluable OS/2 utility products, and money-saving coupons for product upgrades. The Productivity Pack includes: DeskMan/2, DCF/2 Lite, The Graham Utilities*, Relish v2.12, & CPU Monitor Plus*.

*(special version created exclusively for this package)

DEV50 MSRP \$99.95 Save \$25.95

dbfREXX & dbfLIB

by dSoft Development

New! Version 2 Includes Support for NDX & NTX Files!

dbfREXX:

This dynamic link library provides simple, affordable database management from your OS/2 REXX programs. Now accessing dBase files is unimaginably easy.

DSF25 MSRP \$99.00 IB \$95.00

dbfLIB:

Now your C/C++ programs can access dBASE files! dbfLIB provides a consistent set of API's across OS/2, DOS, Windows, Windows NT and Windows 95.

DSF22 MSRP \$195.00 IB \$159.00

4OS2 w/ 4DOS

by JP Software

Open a new world of command line computing with full command history, aliases, enhanced batch files and powerful new file processing options. Includes 4DOS for a consistent interface in OS/2 & DOS sessions.

JPS11 MSRP \$89.00 \$72.00


TENSOR

by Advanced Idea Machines

Looking for a game you can play in minutes or for hours? TENSOR is the game for you! It challenges you like no game since Tetris. Both your ability to find spacial relationships and quick reaction time is challenged.

AIM10 MSRP \$39.95 \$34.95

Visit our web site:
<http://www.indelible-blue.com/ib>

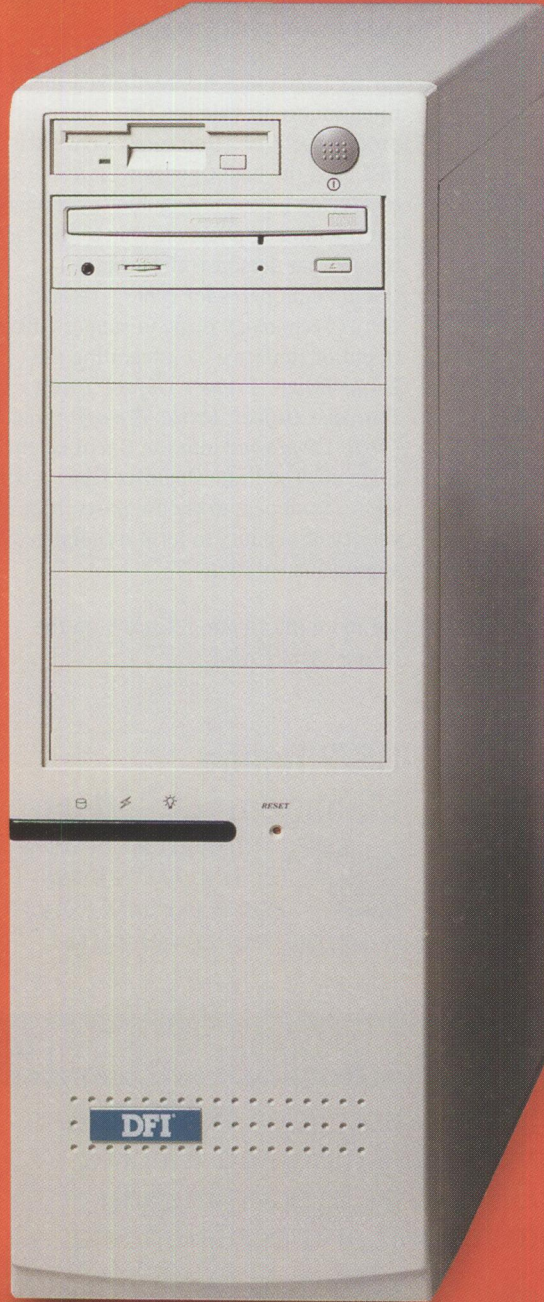
THE SINGLE SOURCE FOR OS/2 SOLUTIONS CALL FOR NEW 64 PAGE FULL COLOR CATALOG!

1-800-776-8284

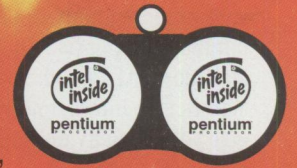
OS/2 and IBM are registered trademarks of IBM Corporation. All other trademarks belong to their respective owners. Prices are subject to change. This ad composed on a PC running on OS/2 Warp, using native OS/2, DOS and Windows apps.

Circle #21 on reader service card.

333MHz* OS/2[®] Performance



Looking for blazing performance? DFI's Doubleshot™ 166 is fast . . . really fast. Two 166MHz Pentium® processors and **preloaded OS/2 SMP** pump this fast iron. One of the few systems that can run two of the most powerful CPUs in the world, the Doubleshot 166 smokes. **Doubleshot Dual CPU**



Doubleshot 166

Two 166MHz Pentium processors
VLSI Wildcat chipset - fastest available
512KB pipeline burst cache
1.6GB hard drive
32MB 60ns EDO RAM
Six speed CD-ROM
4MB PCI video card
28.8 Kbps faxmodem
3.5" floppy drive
PCI/ISA slots
101 keyboard, PS/2 mouse
Tower case - 10 drive bays
OS/2 SMP 2.11 Workstation preloaded
next upgrade free when available
1 Year On-Site Warranty



Call **800-808-4334** for blazing speed
and a free Doubleshot glass

DFI-USA (916) 568-1234
<http://www.dfiusa.com>

DFI-NE (908) 390-2815

DFI-SE (305) 477-1988
<http://www.dfise.com>

DFI[®]

*Did you know that a 166MHz Pentium is actually running at about 166.66MHz?
So two of these give you about 333.3MHz of performance.

Doubleshot is a trademark and DFI is a registered trademark of Diamond Flower Inc. Pentium is a registered trademark of Intel Corporation. OS/2 is a registered trademark and the Ready for OS/2 mark is a trademark of International Business Machines Corp. Specifications subject to change without notice. © 1995 Diamond Flower Inc.

Circle #23 on reader service card.

Stochastic screening minimizes annoying bands and distracting patterns.

A high-speed RISC processor delivers speeds up to 3 ppm (pages per minute) for full color and 12 ppm for monochrome. The Opra C is engineered for high volumes—up to 15,000 pages per month duty cycle. RAMSmart, its advanced memory management feature, ensures it prints smoothly with its standard 8 MB of RAM. (If you require more memory, you can purchase upgrades up to 64 MB.)

PostScript Level 2 and enhanced PCL 5 with color emulations standard give you compatibility with virtually any software application without purchasing expensive options. It supports DOS, Windows, Macintosh, AIX/6000, and UNIX. Two MarkNet XL internal network adapter slots give you virtually unlimited connectivity support.

MarkVision, Lexmark's network print management utility, lets you control, configure, and monitor job statistics of every Opra printer on your network. You can set up the Opra C to receive print jobs from virtually any workstation on your local area network. You'll get real-time alert notifications and remote printer management capabilities. You can even reconfigure the printer and remotely update its flash or hard-disk memory option. Job accounting is easy, too—you can print out lists of user names, printer locations, job duration, capabilities used, number of pages printed, and the nature of any problem encountered, then export the data to a Lotus 1-2-3 or Excel spreadsheet.

For more information, circle 4 on the reader service card.

New Cirrus Technology Products and OpenDoc Support



Cirrus Technology, an OS/2 software developer, offers **Unite Storer**, a hierarchical storage management product supporting optical and CD-recordable jukeboxes.

The SOM-based Unite Storer provides native OS/2 device driver support for a variety of SCSI-based storage devices. It includes a complete set of 32-bit APIs to help you develop flexible and efficient jukebox management applications. Unite Storer can be integrated into development environments such as IBM's VisualAge. Through its graphical user interface, you can start, control, tune, and stop Unite Storer, in addition to performing administrative functions such as importing, labeling, and exporting platters. Dynamic, user-defined tuning parameters are also available for optimizing performance.

Unite Storer takes full advantage of OS/2's multitasking capabilities to manage thousands of store and retrieve requests while simultaneously reading and writing to all drives in the jukebox. Its multi-threaded design can burn CD-recordable discs while under heavy retrieval loads.

Cirrus also released **UniteLite Version 2.0**, personal imaging software for OS/2 Warp based on IBM's Workplace Shell model. New features and productivity enhancements include:

- Support for highlight and pen annotations
- Skew detection and auto-deskew

- Fujitsu ScanPartner 10 support
- Multiple page TIFF file support
- Button bar interface for easier use
- Ability to attach images to cc:Mail messages

UniteLite 2.0 is a personal imaging system for file and folder management. With UniteLite 2.0, you can scan, view, and organize your bitonal documents and color photographs right from the desktop. This office tool enables you to easily get control of all your documents.

Cirrus Technology has reaffirmed their intention to provide the Unite software products as **OpenDoc** component parts. IBM used the beta OpenDoc Unite Objects to develop a powerful multi-vendor sample OpenDoc application. Due to the success of that demonstration and as OpenDoc prepares for general release, Cirrus Technology plans to support IBM's OpenDoc initiative by integrating the **Unite Image Viewer Object**, **Unite Scanner Object**, **Unite Storer**, and **Unite CD-Maker** into the OpenDoc environment. The Unite OpenDoc Objects will allow developers to choose parts from a variety of vendors to form a complete application solution.

For more information, circle 5 on the reader service card.

OS/2 Image Management Software



SYNTEGRATION's newest product, **TRAVELING WORKPLACE**, is a useful tool for backing up, restoring, creating, and

Customized OS/2 Modem Software to handle your mission critical needs.

Whatever you need for large installations, we will exceed your expectations with customized RhinoCom modem software, installation and training. Multi-user licenses, networked modems, BBS, FAX, telecommuting, unattended data warehousing, local / remote routing and conferencing.

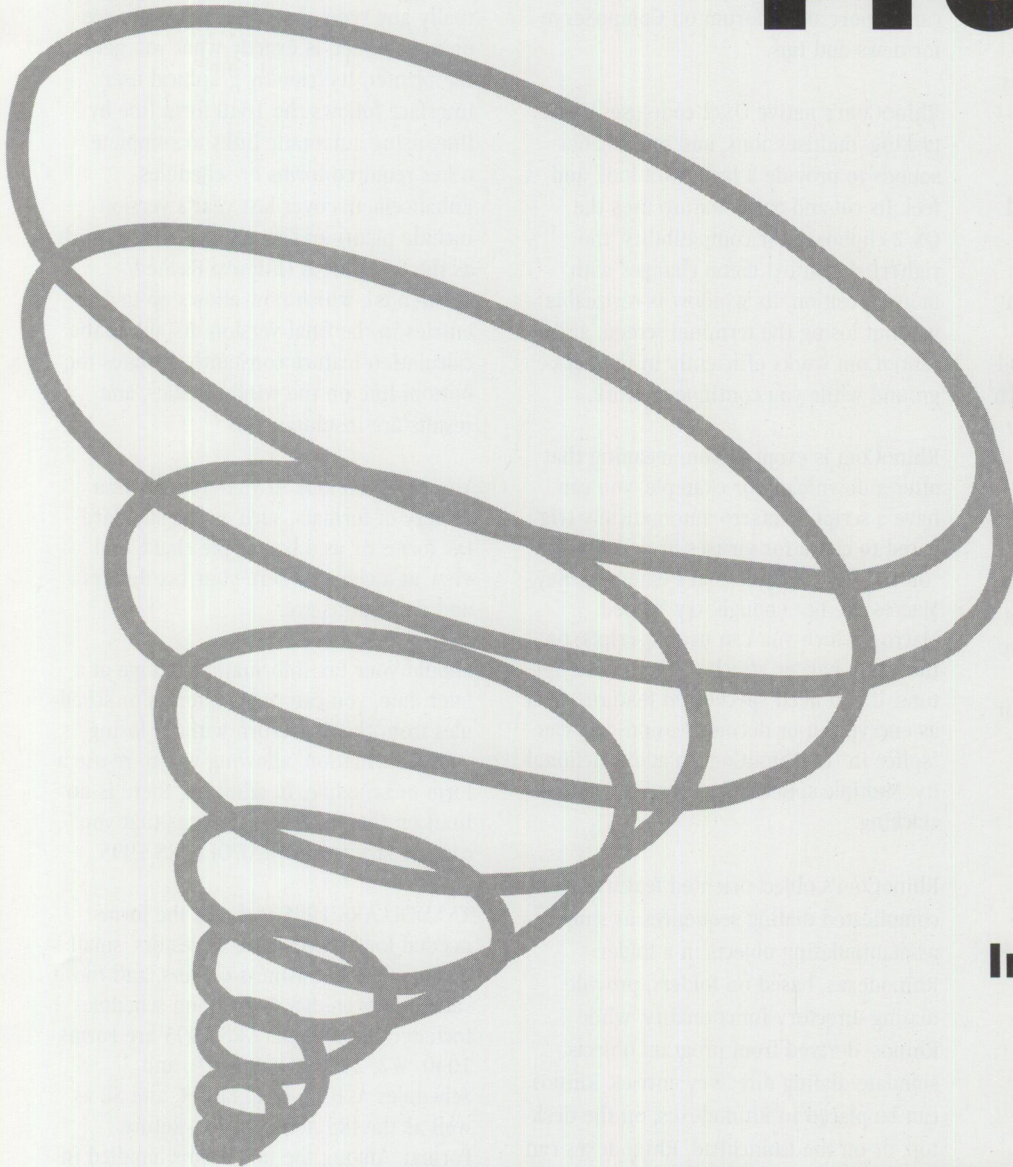
RHINOCom
Reliable OS/2 Communications

Call Today 1-800-234-4546.
From \$5,000. BBS (301) 596-5618
Internet: www.rhintek.com/rhintek
CompuServe: go rhintek

Rhintek
Computer Engineering
Developing System Software since 1977.

Circle #7 on reader service card.

We Have The Largest Selection of **Micro Channel Products**



SYSTEMS
SOUND CARDS
MPEG CARDS
VIDEO ACCELERATORS
SERIAL/PARALLEL PORTS
GAME PORTS
CD-ROMs
SCSI ADAPTERS
NETWORK CARDS
IDE AND EIDE
DATA/FAX MODEMS
EMULATION CARDS
INDUSTRIAL CONTROLLERS
MEMORY UPGRADES
IMAGE CAPTURE CARDS

Call 1-800-GET-MCDA

The MCDA is an independent worldwide organization dedicated to serving developers and users of Micro Channel-based products. Our services include product sales, specifications, seminars, technical support, product catalogs, and newsletters.



Call us to receive our monthly newsletter, or visit us on the Internet at
<http://microchannel.inter.net/microchannel>

169 Hartnell Avenue, Suite 200, Redding, CA 96002 Tel (916)222-2262 Fax (916)222-2528

Circle #24 on reader service card.

managing OS/2 workplace images that can be saved to either a floppy drive, local hard drive, or network file server.

TRAVELING WORKPLACE is fast and offers network administrators many customization options. It backs up a workplace in about 15 seconds and restores it in 25 seconds, without requiring you to shut down and reboot to activate the restored workplace.

This product automatically saves the files that define the settings and icons on an OS/2 desktop, plus lets you select and save additional files with the workplace image. These files will then be recovered when an archive set is restored.

With TRAVELING WORKPLACE, users can back up their own desktops and restore them on other workstations or create and maintain up to 36 online desktops, which they can then switch between. Switching between desktops is much faster than restoring.

You can use TRAVELING WORKPLACE to maintain a different desktop for each user accessing a workstation or to configure a workstation that limits access to sensitive applications. You can also set up and remotely manage different workplaces for users or user groups.

For more information, circle 6 on the reader service card.

Communications Package for Presentation Manager



RhinoCom, from Rhintek Computer Engineering, is a communications package developed for Presentation Manager that lets you customize communications sessions through REXX scripting and powerful macro capabilities. This package emphasizes reliability, versatility, and ease of use.

You can control your communications sessions like other programs in OS/2. You can assign macros to keystrokes, key combinations, the mouse buttons, or variable names for use in scripts. You can attach a fully configurable status box/button bar

to the terminal window or leave it free-floating on the desktop.

Its fully implemented context-sensitive help, online "how to" reference, and interactive "Coach" tutorial aid in ease of use. RhinoCom comes with a manual and a hypertext tutorial explaining basic object tasks. If you can't find an answer in the documentation, Rhintek offers free technical support and a forum on CompuServe for news and tips.

RhinoCom's native OS/2 code uses multitasking, multisessions, and multimedia sounds to provide a true OS/2 look and feel. Its cut-and-paste feature uses the OS/2 clipboard for compatibility, the right-click context menu changes with mouse position, its window is resizable without losing the terminal screen, and RhinoCom works efficiently in the background while you continue to work.

RhinoCom is event-driven, meaning that after a download, for example, you can have a script or macro automatically triggered to check for viruses or decompress "on the fly." If RhinoCom's extensive Key Macros are not enough, try Named Macros, which you can use in scripts, on the menu bar, or attach to the mouse buttons. If you need specialized features such as encryption or decompression, you can "splice in" to RhinoCom to add functionality. Multiple splices allow for easy module stacking.

RhinoCom's object-oriented features make complicated dialing sequences as simple as manipulating objects in a folder. Rhinodexes, based on folders, provide dialing directory functionality, while Rhinos, derived from program objects, simulate dialing directory entries. Rhinos can be placed in Rhinodexes, on the desktop, or on the LaunchPad. Rhinodexes can batch Rhino connections—just once, or until connected.

For more information, circle 7 on the reader service card.

Tax Software for OS/2



TAXDOLLARS 1995 from BT&T Consumer Technology, the only tax planning and preparing software available for OS/2

users, eases tax planning and preparing returns through its intuitive, easy-to-understand format. Even if you're using a tax planner or CPA, TAXDOLLARS 1995 saves valuable time and reduces consulting fees.

Specifically designed to follow Form 1040 and the most common forms, this program prints from a single diskette on virtually any type of printer; what you see on your screen is exactly what you get on the printer. Its friendly graphical user interface follows the 1040 form line by line, using automatic links to complete other required forms or schedules. Enhancements over last year's version include plenty of space in each form, such as the Schedule B (Interest Earned/Dividends), which now allows up to 100 entries in the final version. Its automatic calculation feature constantly updates the bottom line on the window titles, and results are instantaneous.

You can even look at your returns in a variety of formats, such as the standard tax forms or as a bar or pie chart, and view at a glance where your tax dollars and deductions are.

Should your tax information change at a later date, you can remove forms or schedules from the tax return without losing your information, allowing you to re-use a form or schedule. In addition, there is no limit on the number of returns that you can prepare using TAXDOLLARS 1995.

TAXDOLLARS 1995 includes the forms needed for homeowners, investors, small or home-based business owners, and those seeking tax credits. Forms and schedules included in TAXDOLLARS 1995 are Forms 1040, W2, 2106, 2441, 8829, and Schedules A, B, C, D, E, EIC, R, and SE as well as the IRS Accepted Tax Return Format. Among the new forms handled in this version are 1099-INT, 1099-DIV, etc. In addition, state tax calculations are included for California, Illinois, New York, and North Carolina.

To help you plan or maximize your tax situation, TAXDOLLARS 1995 also includes tax tips and IRS statistics. Its 1040 PC direct deposit feature lets you receive your refund directly into your bank account.

For more information, circle 8 on the reader service card.

New Language Based on HTML



Volant Corporation announced *htmlscript*, an easy-to-use language for Web Servers.

Browser independent, *htmlscript* allows you to develop powerful interactive Web pages in a server safe environment.

The product installs as a standard CGI application and provides multiple administrative controls. You can freely mix HTML and *htmlscript* tags in your files. Once executed, *htmlscript* achieves browser independence by operating as a pre-processor, outputting only HTML code. The *htmlscript* tags provide the capability to do logic, calculations, and file input/output operations.

Special licensing, including OEM and site licenses, is available. The software is available for most UNIX and UNIX-like systems.

For more information, circle 9 on the reader service card.

Security Software and Services from IBM



As part of its long-standing commitment to data security, IBM offers a broad range of I/T security products and services designed to protect the enterprise from intrusion.



Emergency Response Service

IBM's *Emergency Response Service* for commercial businesses throughout the world provides swift, expert incident management skills to clients during and after electronic security emergencies. The emergency response team specializes in electronic disasters that affect data processing capabilities.

Available to customers on a subscription basis via IBM's Integrated Systems Solutions Corporation (ISSC), this global service periodically checks customers' networks and can act as an extension of clients' I/T staffs. In the event of a network break-in, the team helps customers detect, isolate, contain, and recover from unauthorized network infiltration. They are on call around the world 24 hours a day, seven days a week.

Customized Weakness Detection Kit

IBM's *Customized Infiltration Tool Kit* is a sophisticated set of tools that detect security weaknesses in clients' Internet connections. With these tools, IBM can probe the subtlest weaknesses that the most sophisticated hackers might try to exploit. These tools exercise network connections that go beyond the capabilities of most existing tools on the market and are customized to match clients' specific network configurations.

Advanced Firewall Security

IBM offers the *Internet Connection Secured Network Gateway*, formerly known as the NetSP Secured Network Gateway. The firewall supports AIX 4.1.3 and operates with the popular RISC System/6000 workstation. It contains an encrypted IP tunnel that encodes data from one firewall to another using DES, the Data Encryption Standard invented by IBM more than 20 years ago, and Commercial Data Masking Facility (CDMF), an exportable encryption technology used outside of North America. The IP tunnel and key distribution is one of the first that is based on the latest IETF specifications, providing the most advanced technology for firewalls currently available.

The Internet Connection Secured Network Gateway also includes remote administration and an alarm capability that allows you to receive alerts triggered by certain errors or other security violations.

Secure Web Servers and Browsers

The IBM *Internet Connection Secure Web Servers* for the OS/2 and AIX platforms and IBM's Internet Connection Secure WebExplorer for OS/2 Warp are now available. These secure Web servers and browser use the industry standard protocols Secure HyperText Transfer Protocol (S-HTTP) and Secure Sockets Layer (SSL) to provide secure methods for conducting commerce over the Internet, including public key data encryption technology and digital signatures.

For more information, see the IT security home page at <http://www.ibm.com/security/> or call (800) IBM-3333, extension IE299 or 1 (520) 574-4600, extension IE299 worldwide.

Full-Featured SQL Database



Sybase, Inc. recently released *Sybase SQL Anywhere Version 5.0*, its full-featured

database optimized for the emerging "new workplace." Driven by the need to extend the IT-enabled workplace from traditional environments supported by centralized IS to anywhere people work and anywhere business transactions occur, new workplace applications are being deployed across a broad range of environments in Fortune 1000, medium, and small businesses, including departments, branch offices, mobile PCs, and customer sites. These applications frequently complement enterprise systems by giving widely distributed and mobile users access to corporate data.

The new version of Sybase SQL Anywhere, formerly known as Watcom SQL, is a member of the Sybase System 11 product family and features enhanced interoperability and compatibility with Sybase SQL Server 11. New in this release are Transact-SQL compatibility support, Sybase Replication Server and Open Client interoperability, Sybase SQL Remote replication technology for occasionally connected users, and the new SQL Central GUI administration tool.

Sybase SQL Anywhere addresses the needs of users in the new workplace by offering the functionality of a true client/server SQL database that can be transparently deployed, easily managed, and economically maintained. SQL Anywhere is available on most popular PC platforms. Its compatibility and interoperability with Sybase SQL Server enables the System 11 product family to provide a total solution, from mobile computing to large-scale, high-performance applications.

Sybase SQL Anywhere features SQL Remote, which provides replication capability optimized for occasionally connected users. This is significant in applications such as field force automation systems where a large number of remote users periodically dial in to a server to upload transactions and synchronize their local database with the server. SQL Remote is based on a hierarchical publisher/

subscriber architecture optimized for large communities of subscribers and supports the Sybase EMS and Microsoft MAPI messaging systems. SQL Remote provides end-user transparent centralized administration, which is a key feature required for simple, widespread deployment. SQL Remote also delivers the ability to regenerate subscriber databases in their entirety, transparently providing the reliability of centralized backup for remote users.

For more information, circle 10 on the reader service card.

Lotus Notes Data Integration Tool



Percussion Software, a leader in Lotus Notes tools, offers a second generation advancement that provides Notes users with quick, accurate access to corporate data for decision support. **Notrix Composer Live!** provides instant bi-directional query and update functionality between a Lotus Notes front-end and a variety of relational databases, allowing Notes users to maintain referential integrity between those databases.

Notrix Composer Live! allows real-time manipulation of data residing on a corporate relational database regardless of the Notes application being used. Until now, integrating data between Notes and relational databases required replicating the data between the Notes front-end and the database and synchronizing events to ensure data integrity.

With Notrix Composer Live! Notes-based information workers can query and update the data warehouse without leaving the Notes front-end, without building an application to achieve the connectivity and integration, and without struggling with the synchronization and contention issues associated with first generation products.

Regardless of the Lotus Notes application accessing the external database, the Notes user fills in a Live! form within Notrix Composer to establish the link between Notes and the transactional database. Once the link is created, users can retrieve designated fields from the external database and display them in their Notes environment.

Data obtained from an external database via Notrix Composer Live! need not be stored in Notes but can be dynamically saved to the external database. With Notrix Composer Live! new applications that combine transactional and document based information can be delivered to multiplatform clients without changes to the Notes application.

For more information, circle 11 on the reader service card.

IBM Virtual World Links Social Computing



IBM has made getting to the information on the Internet more exciting by allowing users, using virtual world technology, to converse, browse historical libraries, and gather information on computer products and services.

IBM Virtual World, a three-dimensional limited demonstration environment, lets visitors search and surf through Web sites by simulated 3-D navigation—entering buildings, strolling down hallways, and conducting text-based chats. People entering the IBM Virtual World select one of six avatars to represent them in cyberspace. The IBM Virtual World adds a third dimension to the Internet, making the experience more lifelike and providing a place for visitors to go for both information and entertainment.

The IBM Virtual World is available, at no cost, from the IBM Software Home Page at <http://www.software.ibm.com/software/virworld.html>. Users can download the software after completing an online registration form and accepting license terms and conditions.

This first version of the IBM Virtual World focuses on the following three locations:

- **The Community Forum**—IBM's center for social computing, the Forum comprises three chat areas where people can meet for open discussion. Chat areas are divided into rooms by industry, user group, and geography.
- **The Los Angeles Public Library Photo Gallery and The Historical Library**—On display in the Gallery are eight digitized photographs from the "Shades

of L.A." collection depicting life in Los Angeles in the early 1900s. The Historical Library allows users to search for and view rare and valuable documents.

- **The Solution Center**—Potential end-users and businesses can learn about IBM products and services before making buying decisions. The Center represents 13 IBM solutions, including database management, application development, and Lotus software, along with IBM hardware, networking, and services. By clicking on a solution, users will be automatically linked to a related Web page for additional information.

For Internet users, IBM offers complete information about the company, products, services, and technology on the World-Wide Web. The IBM Home Page is at <http://www.ibm.com>. The fastest, easiest way to get information about IBM software is to go to the IBM Software home page at <http://www.software.ibm.com>.

Book Answers Frequently Asked Questions About OS/2 Warp



Find answers to the most frequently asked questions about OS/2 Warp with **IBM's Official OS/2 Warp FAQs** by IBM software experts Michael Kaply and Timothy F. Sipples, with contributing editor Bradley Kliever.

Published by IDG Books, this IBM Press book includes answers straight from IBM's technical support database. It clearly explains hardware concepts (such as DMA and interrupts), then provides advice on OS/2 Warp installation, configuration, and hardware selection. **IBM's Official OS/2 Warp FAQs** is one of the first books to address OS/2 Warp Connect and its networking features. The authors also discuss the OS/2 Warp BonusPak and give detailed information on how to use OS/2 Warp with practically any Internet service provider.

This book includes the FAQ Pack, a CD-ROM containing a complete online version of the entire book. The CD-ROM includes extra device drivers, fixes and patches,

hotlinks to popular Internet sites, bitmaps, demonstrations, and free software. For OS/2 Warp novices and power users alike, *IBM's Official OS/2 Warp FAQs* is the perfect companion to IBM's best selling software.

Other IBM Press books include:

Official Guide to Using OS/2 Warp

ISBN 1-56884-466-2-SR28-5668

OS/2 Warp Uncensored

ISBN 1-56884-474-3-SR28-5880-paper

OS/2 Warp and PowerPC

ISBN 1-56884-458-1-SR28-5881-paper

OS/2 Warp Internet Connection

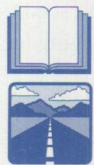
ISBN 1-56884-465-4

The Warp Book

SR28-5785

IBM's Official OS/2 Warp FAQs: ISBN 1-56884-472-7. For more information, circle 12 on the reader service card.

Book on Internet File Formats



Internet File Formats from the Coriolis Group covers the file formats you'll encounter when using the Internet (and other applications). Author Patrick Kientzle, editor with *Dr. Dobb's Journal* and author of *The Working Programmer's Guide to Serial Protocols*, explains the major file formats, how to use them, and where to find information and programs on the Internet for working with them.

You'll learn how to transfer files, how file compression techniques supported by the Internet work, the most efficient way to send big files, how to convert Internet files from one format to another, the best places on the Internet to find different types of files, and how to use Internet file transfer programs such as FTP.

The book begins with information on doing general research on the Internet and the tools that are available. It then describes, in six sections, the major kinds of files you'll work with:

- Text and document formats—HTML, SGML, TROFF, PostScript, and PDF, among others

- Graphics—GIF, PNG, TIFF, JPEG, and VRML
- Compression and archiving—ARC, TAR, ZIP, Compress, GZIP, SHAR, ZOO, and Stuffit
- Encoding—UUEncode, XXEncode, MIME, BtoA, and BinHex
- Sound—WAVE, AIFF, and AU
- Movie—AVI, QuickTime, and MPEG

This book also includes a CD-ROM with helpful tools for working with files. You'll get player programs, compression and decompression utilities, file conversion utilities, file transfer software, and tools to help you create HTML documents.

Internet File Formats: ISBN 1-883577-56-X. For more information, circle 13 on the reader service card.

IBM LAN Server Sourcebook: How to Connect Your Business at Warp Speed



For those of you who already have a LAN installed or who are considering installing a LAN using IBM OS/2 LAN Server, *IBM LAN Server Sourcebook: How to Connect Your Business at Warp Speed* will help you get started. Authors Pat Scherer and Charlie Brown, members of IBM's LAN Server development team, bring their expertise to you, providing you with the information you need to install and use a small to medium size LAN.

Published by John Wiley & Sons, the goal of this book is to help you plan and set up a network that will fit your needs. The first five chapters cover preliminary information—what you should consider before setting up your LAN, including LAN concepts and terminology, LAN Server itself, considerations for connecting your business, and planning advice for saving time and money.

The remainder of the book discusses installing and configuring LAN Server and requesters, LAN Server administration, LAN Server commands, utilities, and

productivity aids. A chapter providing troubleshooting tips will help you keep your LAN running smoothly, and appendices provide additional resources for useful information.

IBM LAN Server Sourcebook: How to Connect Your Business at Warp Speed: ISBN: 0-471-13170-9. For more information, circle 14 on the reader service card.

Way More Free Stuff From the Internet



Way More Free Stuff From the Internet, from the Coriolis Group, follows in the footsteps of its predecessor *Free Stuff From the Internet* by bringing you new fun, educational, and useful stuff you can find on the Internet. Author Patrick Vincent has listed hundreds of interesting sites by category (e.g., food and cooking, games, history, humor, law). His book points the way to plenty of helpful, fun software and information on the Internet.

Did you know you could have Caller ID for Windows? By assigning specific .WAV files to different incoming calls, you'll know who's calling by the sound of the ring. How about documenting your family history with a genealogy application? You can gather, store, and view information about your ancestors with this program. If you have any foreign travel planned, you'll want to go to the Foreign Languages for Travelers Web page and brush up on a few helpful words and phrases. There are lists of them translated in more than a dozen languages, and audio is included so you can hear how the words are pronounced.

Not all of the sites listed in this book are for adults. A chapter devoted to children highlights places of interest they will enjoy. From a money-tracker to a diary to educational material, children can benefit from much of what's available on the Internet too.

Way More Free Stuff from the Internet: ISBN 1-883577-50-0. For more information, circle 13 on the reader service card.

Seton Hall Students Lead the Way From the Wireless to the Real World

By Patrick Karle

"The classroom of the future" is reality right now at Seton Hall University in South Orange, New Jersey. Patrick Karle details the partnership between Seton Hall and IBM's Microelectronics Division, resulting in a state-of-the-art infrared wireless classroom for a group of high-potential students.

When Ken Stelzer realized he and a friend were lost on the rural back roads of Upstate New York over Thanksgiving break, he didn't panic. Stelzer simply called the county sheriff's office on his friend's cellular phone and asked them to fax a map to his trusty IBM ThinkPad.

"We weren't really lost," said Stelzer, a 17-year-old freshman in the W. Paul Stillman School of Business at Seton Hall University in South Orange, New Jersey. "We were trying to find our way to the mall and took a wrong turn." The sheriff's office faxed a map to Stelzer's ThinkPad loaded with WinFax and an IBM PCMCIA (Personal Computer Memory Card International Association) internal DAA 14.4 Kbps data/fax modem. "When the map came up on the screen, it even had a 'you are here' arrow on it. I guess we missed the turn about five miles back," Stelzer said.

Although faxing a map is a small task for IBM mobile technology, the students had successfully demonstrated how to use IBM PCMCIA data/fax modems in an analog-to-digital communication scenario.

Stelzer learned to use IBM mobile technology in classes taught in Seton Hall's Stillman School of Business' new center for leadership studies. He is one of the first 18 freshmen business majors chosen this past fall to participate in the center's Leadership Program, a cohort of men and women with high potential, hand-picked for their academic and life achievements.

Dr. Karen Boroff, director of the center for leadership studies, said the students

are a diverse group, with high SAT scores, bilingual skills, and significant leadership achievements from high school. "In addition to English, they speak French, German, Spanish, Chinese, Circassian, Russian, Ukrainian—even Latin," she said.

The group includes a National Merit Scholarship finalist, a student who spent six months living in Beijing, the captain of a football team, a black-belt in karate, several class valedictorians, and a student who auditioned for TV's *Star Search*.

A Technology Partnership

IBM Microelectronics Division and the Stillman School of Business began their partnership in December 1994 to test PCMCIA wireless technology applications in a classroom environment.

IBM supplied each student and professor in the Leadership Program with an IBM ThinkPad 360 CSE with 8 MB RAM and an Intel 486 DX processor. The ThinkPads are loaded with Microsoft Windows 3.11, Novell PerfectOffice, and Lotus Notes.

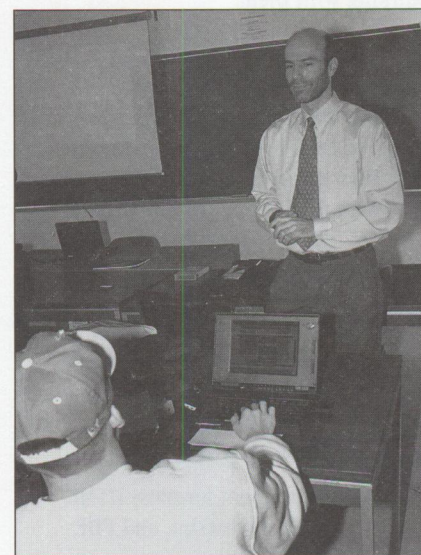
Each ThinkPad can be equipped with an IBM PCMCIA Infrared (IR) Wireless LAN Adapter for connection to an IR Wireless LAN. IBM Microelectronics also supplied students and faculty with IBM PCMCIA internal DAA 14.4 Kbps PCMCIA data/fax modems for communication over phone lines.

Boroff said the students carry their ThinkPads everywhere and use them extensively in classes, including quantitative methods for business, management information systems (MIS), freshman composition, and oral communication.

Students have found various uses for the technology. They use presentation graphics for their speeches in oral communication, they are learning hypertext markup language (HTML) to design their own home pages on the Internet, they sit up all night "surfing the net," and they e-mail each other and their teachers. They use e-mail in class to send notes over an IBM PCMCIA IR Wireless LAN.

John Shannon, dean of the Stillman School, said the Leadership Program was designed to teach high-potential business majors management skills for the 21st century. He said partnering with IBM ensures that Seton Hall has the latest computer technology, service, and training for faculty and students.

"The ultimate goal of mobile computing in the Leadership Program is to make the technology part of the students' thinking and decision-making process," Shannon said. "With IBM, students and faculty all



Dr. Rob Weitz uses IBM ThinkPads and IBM PCMCIA IR Wireless LAN technology to teach current business theory, practices, and software applications.

have the same reliable hardware and software, and they all know how to achieve the same results."

Getting "Un"-wired

Last summer, technicians from IBM Microelectronics Division's Toronto Lab installed four IBM 8227 IR Wireless LAN access points in the ceiling of the classroom where the leadership group meets. IBM also installed access points in the business school classrooms and the library for individual or small group wireless networking and/or access to the wired network.

The IR Wireless LAN access points are small black transceiver boxes about the size and shape of a video cassette. These access points communicate with the IBM PCMCIA Wireless LAN Adapters installed in the instructors' and students' ThinkPads, allowing data transfer in the classroom using diffused infrared light instead of a cabling system.

The access points in the ceiling are wired to a server running Novell software, which treats the IR Wireless LAN like the other wired LANs in the business building. A student using a ThinkPad equipped with a PCMCIA Wireless LAN Adapter can access the entire wired world via IR wireless communication technology.

The PCMCIA IR Wireless technology can be used in two ways. On campus, students and teachers can log onto the PCMCIA IR Wireless LAN via their IBM PCMCIA IR Wireless LAN Adapters during class to share printers, disk drives, and data. Between classes or in small group study sessions, it's also possible for students and faculty to log on to the IR Wireless LAN to share data, print documents, access the Internet, or send and receive e-mail.

Off campus, by using their ThinkPads as stand-alone computers or communicating over phone lines via their IBM PCMCIA data/fax modems, students and faculty avoid using on-campus wired computer labs, which can be crowded, especially around exam time.

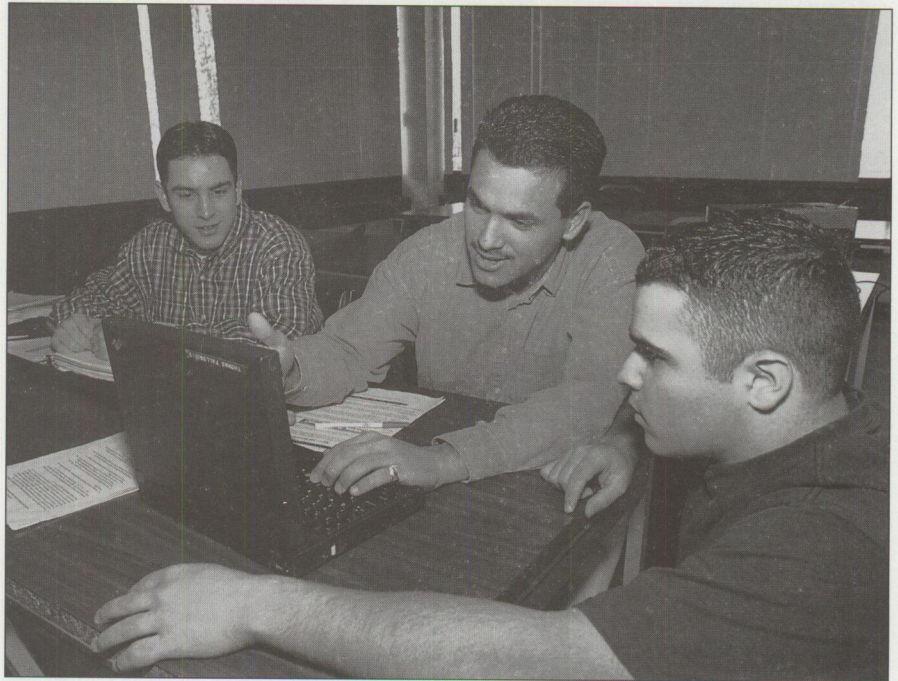
Phillip Parks, a Seton Hall freshman, said the best feature of mobile computing is that you don't have to trudge to the campus through the rain and snow and sit in the lab half the night typing away. "You

can sit home in bed and write your paper," Parks said.

Computing the Benefits

Dr. Yonah Wilamowsky, who team teaches quantitative methods, an advanced statistics class in the Leadership Program, said mobile technology allows students to

focus on math concepts rather than mere computation. Because they are using a statistical analysis software package on their ThinkPads, he said he can give the group larger, more complex, and more realistic blocks of computations than he would give to math students not using ThinkPads.



Between classes, students use IBM PCMCIA IR Wireless LAN adapters and IBM ThinkPads to access the Internet and send e-mail.



IBM PCMCIA IR Wireless LANs allow students to share data and print documents in the classroom.

ThinkPads can be networked on the IR Wireless LAN in class. Students can also use their ThinkPads to perform the computations away from the classroom, so they can spend more time in class analyzing the results.

"Using the ThinkPad changes the math experience," Wilamowsky said. "Students can't get lost in or hide behind the computations. The goal of math class is no longer just getting the right answer," he said. Students learn how to interpret the numbers and explain their ramifications.

Dr. Rob Weitz, who teaches the MIS course, said mobile technology allows instructors to teach students how to use common software packages, then teach them how to apply that technology to business concepts.

He teaches them how to use a remote mail package with their PCMCIA modems to send e-mail, then shows them how e-mail is changing the way people do business. "The next step is the possibility of using Lotus Notes to create our own bulletin boards," Weitz said.

Weitz feels that the IBM ThinkPad and PCMCIA technology is impressive, and teaching the students how to use the technology inspires them to learn. "They are excited about learning a technological procedure or a software package, and that excitement extends beyond the classroom," he said.

A Case in Point

As soon as Miguel Pagan and Tom Palagoudis learned about spreadsheets, they used QuattroPro to set up their weightlifting schedules.

Pagan, who won a Rotary Club Leadership Award in high school, said he knew very little about computing before he entered the Leadership Program, but now he uses spreadsheets for everything. "Spreadsheets are great. You can even use them to keep statistics for a sports team," Pagan said.

A Learning Experience

Shannon believes the partnership with IBM is a good one. "IBM is learning from us and we are learning from IBM. They are innovative. We are innovative. It's refreshing to work with people who are willing to stay so far ahead of the curve," he said.

The IBM connection helps make the Leadership Program, Seton Hall University, and its students more competitive, according to Shannon.

There are many business schools out there, but Seton Hall is the only school in New Jersey that offers to teach students how to use mobile computing. "I don't see how a business major can be competitive in today's job market without mastering mobile computing," Shannon said.

Mobile computing is a culture, a way of life in the New York metropolitan area, where commuting is also a way of life; learning leading-edge information technology job skills is one advantage the Leadership Program offers its students.

Heavily recruited by the largest financial, insurance, and pharmaceutical industries in the world, Seton Hall graduates hold some of the most respected positions on Wall Street. Getting top jobs when they graduate requires leadership in both the wired and the wireless worlds. When graduation comes, these students will be ready.



Patrick Karle, author of many articles on IBM products, is managing director of Patrick Karle Communication, a high-tech marketing communication firm, in Princeton, NJ.

Last year, Patrick won a Public Relations Society of America award for a 1994 international media relations campaign promoting IBM ThinkPads and an STP/Auto Racing Writers and Broadcasters Association Writing award for "OS/2 Times and Scores the 1994 Indianapolis 500" (*Personal Systems* July/August 1994).

You can reach Patrick at PKar1e@MSN.Com.

“Out, Damn Spot!” or How to Rid Your OS/2 Desktop of Pesky Programs

By Mike Norton

Having trouble getting rid of old applications? Want to clean up your INI files? Need to split your EAs? Well, before you resort to Lady Macbeth's removal methods, read on. SofTouch Systems' UniMaint will take care of these pesky problems for you.

Although they are actually two separate entities, OS/2 and the Workplace Shell have become so intertwined in users' minds that they are practically synonymous. The Workplace Shell gives OS/2 its distinct personality. The Desktop is the pride and joy of every OS/2 user.

Utilities have existed for some time for protecting the Desktop. The Workplace Shell, however, has remained as mysterious and intimidating as disk drives once were. UniMaint from SofTouch Systems provides you with a glimpse into the mysterious world of the Workplace Shell and arms you with tools for maintaining and protecting this powerful, integral part of OS/2.

The Workplace Shell works its magic by maintaining invisible associations between objects, for example, between a data file and its associated program. Something every new OS/2 user discovers is that removing applications is no longer a simple matter of deleting the application's directory. To mimic Dorothy's famous quote, “We're not in DOS anymore,

Toto.” Modern applications designed to exploit the operating system's technologies typically create user .INI entries as well as install help files and DLLs. Newer, bolder applications are exploiting the potential of object classes. In this technologically advanced environment, each application should provide for its own uninstallation. Unfortunately, few do.

The Workplace Shell works its magic by maintaining invisible associations between objects. . .

This is where UniMaint, the first uninstaller for OS/2, enters the picture. Unlike Windows uninstallers, which journal applications and are of no use for applications installed before the uninstaller, UniMaint can uninstall existing applications. Simply drag the application's .EXE file to the uninstaller window in the

Tools Folder (Figure 1). UniMaint determines associated help files and DLLs, searches the CONFIG.SYS file, removes system .INI file entries, de-registers proprietary classes, and destroys application directories and files. And it works not only on OS/2 applications, but on DOS and WIN-OS2 applications as well.

This feat is accomplished by using technologies acquired through years of developing system maintenance utilities. UniMaint is more than just the first uninstaller for OS/2; it is a complete package for maintaining OS/2's critical Workplace Shell facilities. Most users are shocked, for example, by the sheer number of obsolete, invalid entries infesting their critical OS/2 .INI files.

These .INI files—repositories of persistent data—suffer chronically from the poor housekeeping habits of many programs and users. Many processes—including OS/2 itself—often don't bother to clean up after themselves. And often a user simply deletes an application directory, unaware of its associated .INI entries. These forgotten entries degrade system performance and, in the worst case, can cause system failure. And because OS/2 .INI entries are often in the more efficient binary format (unlike the Windows ASCII text WIN.INI file),

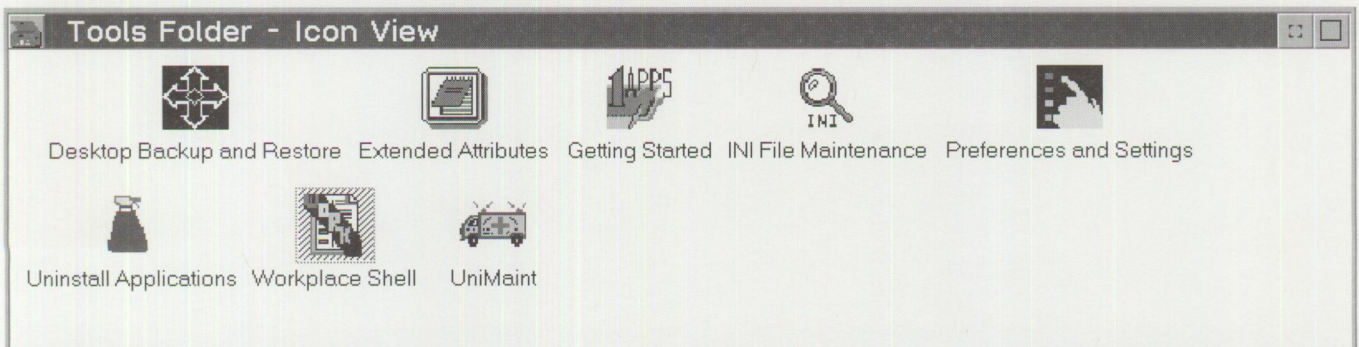


Figure 1. UniMaint's Tools Folder

manual maintenance is generally beyond the technical expertise of the average user.

UniMaint provides a safe, reliable, effective, and automated way to rid your system of these unwelcome pests. This technology has been tested on thousands of OS/2 machines and has proven to be safe and reliable. If your system is mission critical, you cannot afford to be without UniMaint.

Besides repairing .INI files, UniMaint also serves as a full-featured editor for any OS/2 .INI file or extended attributes (EAs). Developers will find UniMaint invaluable for testing their own routines exploiting the Workplace Shell. Users who like to tinker with their system will find numerous uses for UniMaint's .INI file and EA editing features, from copying or splitting EAs to repairing application .INI files. Even the most casual user will receive an invaluable education on the Workplace Shell's inner workings by examining .INI files and EAs with UniSafe, which allows the non-technical

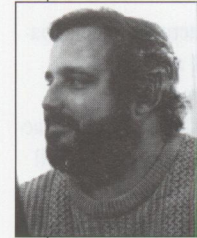
user to safely view this critical system information without any risk of overwriting data.

An ounce of prevention is worth a pound of cure—backups are the best form of prevention. So UniMaint naturally provides facilities for backing up the Desktop and OS/2 .INI files. Backup is as simple as clicking on an icon in the UniMaint Tools folder, or it can be run from a command line or .CMD file. The Desktop and its critical extended attributes as well as the .INI files are zipped into a file that can be restored using a simple REXX script (generated by UniMaint) from OS/2 maintenance mode.

With UniMaint, you can store multiple generations of Desktops with convenient descriptions associated with each. You can back up other critical files, such as CONFIG.SYS and PROTOCOL.INI. You can also port your Desktop to another machine. UniMaint uses the portable backup to compare objects on the source machine with applications installed on

the target machine, then creates the associated object for each file or application present. This process allows UniMaint to recreate Desktops independent of the operating system version, which means Desktops can be ported across versions of OS/2.

UniMaint is available from SoftTouch Systems, 1300 S. Meridian, Suite 600, Oklahoma City, OK, 73108. Voice: (800) 944-3028, Fax: (405) 947-8169. CompuServe: GO SOFTOUCH. Or circle 15 on the reader service card.



Michael Norton is the Workstation Division Manager at SoftTouch Systems, which provides both mainframe and PC software solutions. He has a BA in Philosophy and has written mainframe

manuals, as well as articles in a number of PC publications. You can reach him at mnorton@softtouch.com.

Why SOM?

By Brian Curran

In this article, Brian Curran overviews IBM's System Object Model (SOM) to give you a better understanding of its impact on you, whether or not you're an application developer. After defining SOM in detail, he highlights its technical strengths and significance in the marketplace.

SOM stands for IBM's System Object Model. For most of you, that's probably a cryptic term. So you may be asking the following questions:

- I've heard of SOM, but does it really impact me?
- What's so special about this technology called SOM?
- What benefits does SOM provide me as a software developer?

This article answers these questions, including why SOM technology impacts you, whether or not you are a software developer. Other SOM-related articles in

this issue describe how to use SOM, including:

- Integrating SOM with other languages: VisualAge Smalltalk and VisualAge COBOL—both using the same Valet class library
- Using a feature called Direct-to-SOM that enables a programmer to define SOM objects with C++ instead of IDL
- Using SOM's distributed component, called DSOM (Distributed SOM)
- Glimpsing the future of SOMobjects 3.0 Object Services (*Note:* The information provided in these "future" articles is

from the SOMobjects 3.0 beta and is therefore subject to change.)

The goal for *all* of these SOM-related articles is simple—we want to pique your curiosity and raise your overall awareness of SOM technology. In this issue, we present a relatively small amount of information regarding SOM. Each topic uncovers only the tip of the iceberg. Much more information exists—it just won't fit within the covers of this magazine. Our hope is that after seeing the tip of the iceberg, you will want to discover all of SOM's benefits.

What is SOM?

At the highest level, SOM is a language neutral, object-oriented programming technology for building, packaging, and manipulating binary class libraries that can share objects across address spaces. Whew! I'll elaborate on this definition, overview the technical merits of SOM, and describe why an application developer might use SOM later in this article. First, I want to demonstrate SOM's significance by identifying just a few of today's leading edge technologies that incorporate SOM's benefits.

OS/2's Workplace Shell

The Workplace Shell (WPS), OS/2's user interface, uses SOM technology. Every user interface object on the WPS is a true SOM object, and many of the WPS's dynamic capabilities are provided due to SOM's technological underpinnings.

Workplace Shell Applications and Tools

Many applications are WPS-enabled, meaning that you interact with the application using WPS objects (which are SOM objects). To provide WPS extensions or tools, developers use SOM technology when developing their software.

OpenDoc

SOM, the object model CILabs (Component Industry Laboratories) selected specifically for OpenDoc technology,



provides a packaging technology for OpenDoc components. Larry Loucks, vice president of software for IBM Personal Software Products, states, "SOM just gets us plumbing. We spent a lot of time up front getting out a couple releases of this stuff to make sure we have a good understanding of the infrastructure before we get to the killer, which is component software . . . our approach to that is OpenDoc." as reported by Orfali, Harkey, & Edwards in their book, *The Essential Distributed Objects Survival Guide*.¹

Orfali, Harkey, & Edwards communicate the dramatic potential of SOM: "As an integral part of OpenDoc, SOM will ship with every copy of OS/2 Warp, Macintosh, and AIX. This means SOM will become the first high-volume CORBA [Common Object Request Broker Architecture] ORB [object request broker] . . ." The authors conclude, "It will take some time before programmers get accustomed to the idea that a full-function SOM ORB is shipped with every OpenDoc runtime." It is important to note that the platform support for SOM is much broader than the initial support for OpenDoc, which is a positive statement regarding OpenDoc. With SOM support on additional platforms (such as AS/400, MVS, and others), OpenDoc components will be able to exploit SOM's distributed capabilities and communicate with distributed objects on these additional platforms.

OS/2 SOM-Compliant Products

A scan of recent computer technology magazines revealed a large number of advertisers claiming SOM-compliant programs. Figure 1 lists a few of the products employing SOM technology.

Whether or not you are interested in the ensuing technical section, the emerging technologies and large number of products employing SOM technology hopefully convince you of its importance. Now, let's take a look at why so many applications are employing SOM technology.

SOM's Technical Highlights

Remember that impressive definition of SOM earlier in the article? *At the highest level, SOM is a language neutral, object-*

¹ The Essential Distributed Objects Survival Guide. Robert Orfali, Dan Harkey, Jeri Edwards; John Wiley & Sons, 1996. ISBN 0-471-12993-3.

Product	Manufacturer
JBA Guidelines	JBA International
ChipChat	ChipChat Technology Group
Object Desktop	Stardock Systems
Unite Objects	Cirrus Technology
KidProof/2	Pinnacle Technology
HyperACCESS	Hilgraeve
DeskMan/2	Development Technologies
VX-REXX	Watcom
VisualAge C++	IBM
High C/C++ Toolset	MetaWare

Figure 1. Sampling of SOM Technology Products

oriented programming technology for building, packaging, and manipulating binary class libraries that can share objects across address spaces. Depending upon your perspective, that definition either has dramatic impact or provides very little meaning. Let's examine each part of this definition.

Language Neutral

SOM is language neutral, meaning that SOM class users and creators don't have to use the same programming language. For example, a client application need not be written in C++ to create, manipulate, and delete a SOM object implemented in C++.

SOM does not replace existing object-oriented languages; rather, it complements them so that application programs written in different programming languages can share common SOM class libraries, such as the Valet class library.

Note: SOM's language neutral capabilities are demonstrated in accompanying articles. The Valet and associated classes are implemented using C++, whereas the client programs demonstrated are VisualAge C++, VisualAge Smalltalk, and VisualAge COBOL.

Object-Oriented Programming

Entire books are written on object-oriented programming. Most of these books identify three key characteristics when determining the merits of an object-oriented environment: encapsulation, inheritance, and polymorphism. SOM fully supports all three characteristics.

Many of SOM's original design points were introduced as a better way to package object-oriented technology. All software environments have shortcomings, but SOM solves many of the deficiencies introduced by other object-oriented technologies, such as the fragile base class problem introduced by C++. In addition, SOM combines the strengths of various object technologies into one environment. Like Smalltalk, SOM supports class objects and name resolution method dispatching. Like C++, SOM supports multiple inheritance and provides strong type-checking. If you are interested in object technology, I strongly recommend that you study SOM technology.

Building, Packaging, and Manipulating

When using SOM technology to build, package, and manipulate class libraries, you will circumvent many of the deficiencies introduced by other object technologies. For example, SOM can be used with C++ to provide upwardly compatible class libraries.

You can release a new version of a SOM class, and as long as no changes to the client's source code are required, the client code does not have to be recompiled. For example, let's say you have a brainstorm and can now rewrite your `GrainsOfSandOnBeach()` method with a couple of new instance variables. When you re-release your Beach class library, none of the clients will need to recompile since the public interface remains the same. (This is the way object-oriented programming is supposed to work.)

SOM accommodates changes in implementation details without breaking the binary interface to a class library or requiring client programs to be recompiled. Specifically, you can make the following changes to SOM classes and still retain full backward, binary compatibility:

- Add new methods
- Change an object's size by adding or deleting instance variables
- Insert new parent classes above a class in the inheritance hierarchy
- Relocate methods upward in the class hierarchy

In contrast, most changes to a C++ class library, including adding or removing private instance variables, require client code to be recompiled. Keep in mind that the public interface does not have to change to force a recompilation. The C++ recompilation requirement seems trivial until you consider the following two scenarios.

Scenario 1: Your enterprise solution includes a single C++ class library that contains functions for eight applications located on 3,000 desktops. Your CIO requires that additional information be tracked, mandating a single additional instance variable and associated accessor methods. You now face the unenviable task of updating the class library, recompiling all eight applications, and redeploying 17 MB of code (eight applications at 2 MB per application plus 1 MB for class library) to 3,000 desktops. In reality, the eight applications are each in a different phase of development, and it is highly unlikely that all eight would be deployed at the same time, thereby increasing complexity.

If your class library is implemented using SOM, your workload is significantly reduced. Only the single 1 MB class library needs to be redeployed to all of the workstations. Your existing applications will not need to be recompiled and will work with the new SOM class library. Deploying a single 1 MB class library is simpler and less resource-intensive.

Scenario 2: You are a vendor of a Valet class library and have sold copies to 1,000 application developers. Hypothetically, these application developers build

applications for managing all of the parking garages throughout the world. You now have a new version of the class library that incorporates bug fixes, performance enhancements, and increased functionality from a variety of new classes to support long-term parking.

Let's assume 100 of your clients are leading-edge application developers. They will be the first to extend their applications to support the library's new long-term parking feature. Obviously, the client applications will need to be recompiled to include the enhanced functionality. But what about the other 90 percent of your clients? Will they need to recompile their applications even if they don't need the functionality?

Assuming the Valet class is implemented in C++, the answer is yes. Unfortunately, developers will be forced to recompile their applications, even if their only goal is to obtain the bug fixes and performance enhancements. Of course, they could decide to ignore the update (at a cost of running with known bugs).

Assuming the Valet class is implemented using SOM, the answer to the previous question is no. Recompilation by the other 90 percent of the clients is not required. The same Valet class library will support applications with enhanced long-term parking as well as the existing applications written when the Valet class library did not have the long-term parking feature. SOM's upwardly compatible class libraries provide this support.

Find "SOM" More Information on the World-Wide Web!

You can obtain the complete source code used for the examples in this issue's articles on *Personal Systems'* home page at <http://pscc.dfw.ibm.com/psmag/>. And while you're on the Web, you might want to take a look at some other object technology sites:

IBM Object Technology

<http://www.software.hosting.ibm.com/objects/>

A great starting place for finding information, this site offers platform-specific information, links to IBM partners in object technology, and information on products, education, and services.

IBM Solution Developer Operations

<http://www.austin.ibm.com/developer/objects/>

This site offers developers information on SOM, OpenDoc, and Taligent's CommonPoint, plus information on developer programs, an object technology library, links to related Web sites, and object-oriented news.

SOM Programming Forum (the IBM Software Home Page)

<http://www.software.ibm.com/qa/www/>

Take the opportunity to participate in valuable exchanges with IBM's experts in object-oriented technology. "SOM

Programming with Christina Lau," a December 1995 forum, covers a variety of topics.

SOMObjects Technical Support

<http://www.austin.ibm.com/somservice>

Browse FAQs, update your SOMObjects Developer Toolkit or SOMObjects Workgroup Enabler, submit a question or problem for the technical support team, and obtain CSDs.

Bart's Home Page

<http://goban.austin.ibm.com>

On his home page, Bart Jacob, senior developer with International Technical Support Organization in Austin, offers information on object-oriented projects from the ITSO Center, access to all ITSO Redbooks, and object-oriented Redbooks.

OMG (Object Management Group)

<http://www.omg.org>

Visit the non-profit consortium's home page to find out more about the organization, obtain membership information, and view the list of current members. You'll also find industry information—events, suggested reading, OO technology user groups, plus links to related sites.

Binary Class Libraries

SOM lets you deliver your class libraries in binary format. In layman's terms, this means that as the producer of a SOM class library, you are able to keep proprietary information proprietary. In delivering a SOM class library, you provide two files: the DLL containing your class implementation and the IDL file containing the interface description. With these two files, any individual using a language that supports SOM can instantiate objects based upon the class library or designate the delivered classes as sub-classes and provide specializations! (Yes, our good friend, language neutrality, described earlier, allows a SOM class implemented in C++ to be extended and derived in a completely separate language.)

In contrast to delivering a SOM class in binary format, C++ programmers usually take the lesser of two evils and deliver the source code along with the class library. Few software engineers are accustomed to delivering source code as part of a procedural function library, but this practice is actually common with C++ class libraries. The alternative is too much of an administrative nightmare. If a class library provider chooses to withhold the source code, then the class library provider is making the decision to develop, test, ship, and track an instance of the class library for each of the C++ compilers on the market today. And when you consider the need for maintaining multiple versions for each instance of the class library, the task is even more daunting.

Share Objects Across Address Spaces

Now, let's look at the last part of the SOM definition. In an ideal software vacuum, objects would be ubiquitous and the benefits of object-oriented technologies would be enjoyed by all. Realizing there is no such vacuum, an object system of even minor complexity must cross the dreaded procedural boundary one or more times, whether it's to an operating system-specific service, a communications API, or some other type of legacy interface.

For those who believe in the benefits of objects (or at least who believe in the potential for making profits with objects), the race is on to extend the procedural boundary—one of the reasons why we see such a proliferation of class libraries in today's environments. Initially, there was tremendous emphasis on providing

an object-oriented abstraction for GUI programmers. Disparity arose, however, when elegant user interfaces were being developed and programmers were still coding linked lists. Thus, we now have robust collection classes that should free up the technical resources from coding linked lists. Slowly but surely, the procedural boundaries are being extended to further and further extremities of software systems. The move to distributed objects is a natural progression of the desire to further extend the procedural boundaries. There are two reasons why you will want to share objects across address spaces:

- Objects make sense in client software, so they will also make sense in server software
- Objects on the client should be able to communicate with objects on the server

DSOM, SOM's distribution component, lets you share objects across address spaces, either in the same machine or across the network. DSOM simply adds distributed capabilities to the benefits already inherent in SOM classes. DSOM also conforms with the Object Management Group's CORBA standards. More DSOM information is provided in "Distributing Objects with Distributed SOM" in this issue.

Why Use SOM in Your Application Development?

If you aren't convinced by now, here are my top 10 reasons for using SOM in your application development:

10. To develop a WPS application or tool
9. If you work for an object technology center and keep two object models—one for the C++ folks and one for the Smalltalk folks
8. To understand deficiencies of other OO environments
7. To develop component-based software, specifically using OpenDoc
6. If you want object-oriented code on your clients as well as your large system hosts
5. If you don't want to ship your proprietary source code
4. If you are tired of forcing all of your clients to recompile when you change your class
3. To develop a class library and expand

your market potential beyond a single language, compiler, and platform

2. If you are interested in the promises of CORBA and want to develop distributed object systems
1. If you want a real system object model!

Conclusion

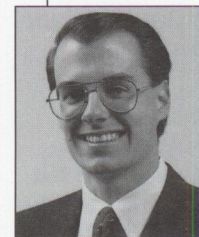
In this article, I've attempted to:

- Introduce SOM technology and show its impact on you, whether or not you are an application developer. With a better appreciation of SOM technology, you can be a more informed purchaser of application software.
- Overview SOM's technical highlights, as well as its technical strengths.
- Assist you in understanding when you should use SOM technology in your application development.

You'll find more concrete details in the remaining SOM articles in this issue. If you are familiar with C++ and Smalltalk, SOM looks different. But, then again, both C++ and Smalltalk looked *quite different* when I first saw each. So keep after it; you should be up and running with a minimal amount of effort.

References

- The Essential Distributed Objects Survival Guide*. Robert Orfali, Dan Harkey, Jeri Edwards; John Wiley & Sons, 1996. ISBN 0-471-12993-3.
- Programmer's Guide, Volume I: SOM and DSOM*. SOMObjects Developer Toolkit. SOMObjects Version 3.0 beta.
- Programmer's Guide, Volume II: Object Services*. SOMObjects Developer Toolkit. SOMObjects Version 3.0 beta.



Brian Curran is a senior marketing support representative for the Personal Systems Competency Center (PSCC). He obtained a BBA and an MBA in Business Computer Information Systems from the University of

North Texas. Since working with the PSCC, Brian has concentrated on OS/2 application development and most recently has focused on IBM's Object Technologies, including SOM and DSOM. He can be reached via e-mail at brian_curran@vnet.ibm.com.

IBM System Object Model— The Wave of the Future (and Now!)

How does IBM's System Object Model (SOM) fit in the object world? What does SOM mean to you as a developer? The technical articles in this issue answer these questions by closely examining SOM's features and framework to show why SOM should be the infrastructure you choose for object development.

This article describes a valet parking object model, which is used by other articles in this issue. Refer to this article when reading the others.

Object-oriented analysis, object-oriented design, object-oriented databases, persistent objects, object-oriented application development . . . objects are everywhere!

"Objects" is certainly the buzzword of the '90s. You've probably heard about, and perhaps have been sold on, the advantages of objects and object-oriented application development. If you wade through some of the objects hype, you'll find some tangible benefits of using objects, including:

- Improved design
- Improved code reuse
- Improved code maintenance

Rick Weaver
IBM Corporation
Roanoke, Texas

Because traditional object-oriented development uses languages such as C++ or Smalltalk, there are some inhibitors. At a high level, SOM solves the problems developers using these languages deal with daily:

- How to share objects among different languages
- How to distribute class libraries in binary format

- How to enhance a class library without forcing the class library users to recompile their client code
- How to distribute and access objects on different platforms

SOM Benefits

With SOM, you can:

- Allow languages, such as Smalltalk, C, and other languages that support SOM, to use your SOM classes
- Develop class libraries and ship only the binary form of the classes, meaning you don't have to ship your source code with a class library
- Change a class and maintain full backward compatibility, meaning that any

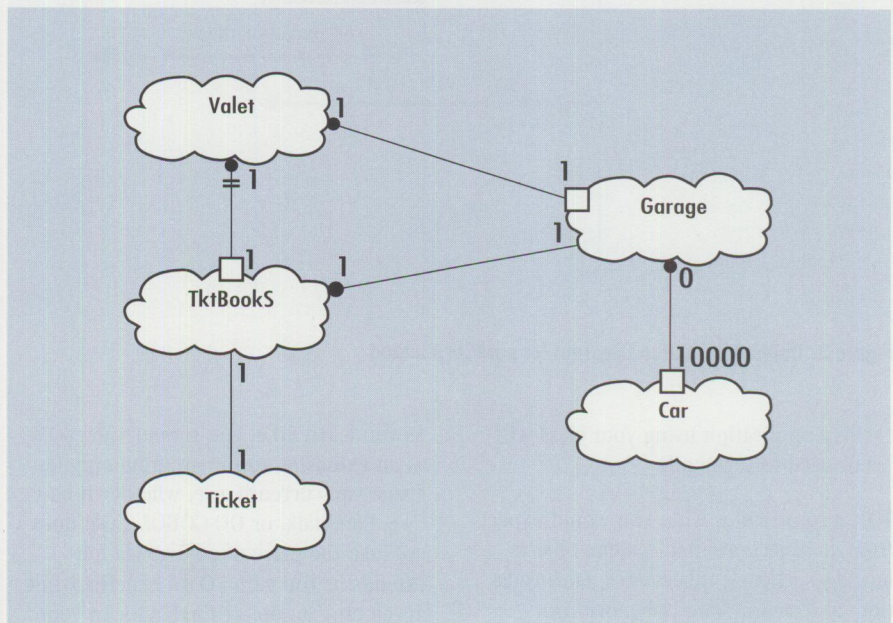


Figure 1. Object Model in Booch Notation

```

/* Valet.idl */
#include <somobj.idl>
#include <Garage.idl>
#include <TktBookS.idl>

interface Car;
interface Valet : SOMObject
{
    attribute string valetName;

    Ticket parkCar(in Car aCar);          /*parkCar method*/
    Car retrieveCar(in Ticket aTicket);   /*retrieveCar method*/

#ifdef __SOMIDL__
implementation
{
    Garage aGarage;
    TktBookS aTicketBookSystem;

    passthru C_xh =    "#include \"Ticket.xh\""
                    "#include \"Car.xh\""
                    "#include \"Garage.xh\""
                    "#include <TktBookS.xh>";

    dllname = "Valet.dll";
    releaseorder : _get_valetName, _set_valetName, _get_valetId,
                  _set_valetId, parkCar, retrieveCar;
    somDefaultInit : override, init;
    somDestruct : override;
};
#endif
};

```

Figure 2. Valet Class Interface Definition Language (Valet.idl)

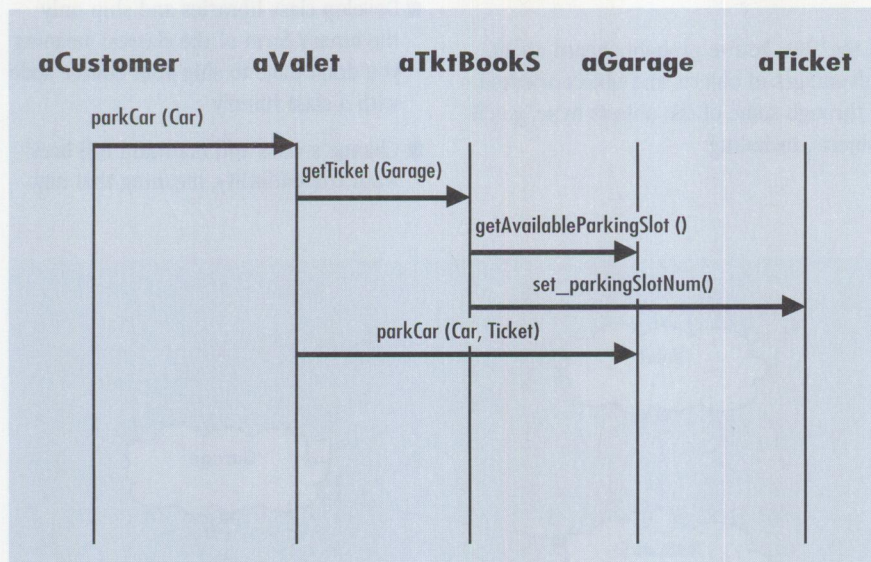


Figure 3. Object Interaction Diagram for parkCar Method

client application using your class will not need to recompile

SOM also provides what you would expect from an object-oriented system: objects, messages, classes, inheritance, encapsulation, abstraction, and polymorphism.

As you learn SOM, you should view SOM as an extension to the programming language you currently use, whether it be C, C++, Smalltalk, or OO-COBOL. SOM does not take the place of a language but extends it. You gain SOM's benefits while developing an object's behavior in your preferred programming language.

The Model

Several articles in this issue use a simple object model for some of the code examples. The object model follows a valet parking system scenario, where cars are parked in and retrieved from a parking garage. The valet parking system works as follows:

A customer drives to a stadium and gives his car to the valet who parks the car in the garage and provides a parking ticket to the customer. Later, the customer returns the parking ticket to the valet. The valet retrieves the car from the garage and returns the car to the customer. All parking is free in the garage, which has a 1,000 car capacity.

The object model (in Booch notation) is illustrated in Figure 1.

The classes used in the valet parking application include: Valet, Ticket, Car, Garage, and TktBookS.

The Garage and TktBookS classes are internal to the Valet, and any clients using the valet parking system are not concerned with these classes.

Valet Class

An attribute is private data that will have SOM automatically generate `_get` and `_set` methods. The Valet class has two attributes:

- valetName—a string
- valetId—a long integer

The Valet class supports the following methods:

- parkCar
- retrieveCar
- _get_valetName
- _set_valetName
- _get_valetId
- _set_valetId

The two Valet methods are the `parkCar` method, which accepts a `Car` and returns a `Ticket`, and the `retrieveCar` method, which accepts a `Ticket` and returns a `Car`.

The *interface definition language* (IDL) is a CORBA-compliant, language-neutral way to define a class in SOM.

The Valet class IDL is shown in Figure 2. Figure 3 shows the object interaction diagram for the parkCar method, while Figure 4 shows the object interaction diagram for the retrieveCar method.

Ticket Class

The Ticket class has two attributes: ticketNum and parkingSlotNum—both of which are long integers. It also has _get and _set methods defined for both attributes.

Car Class

The Car class has three attributes: color, make, and model—all of which are strings. It also has _get and _set methods defined for the three attributes.

Garage Class

The Garage class has one instance variable: garage_lot—a CORBA sequence. An *instance variable* is private data that has no _get and _set methods generated by SOM. A CORBA sequence is simply an array that will be used for storing cars. The Garage class supports the following methods:

- storeCar
- removeCar
- listCars
- getCarCount()
- getAvailableParkingSlot
- removeAllCars()

The Garage class IDL is shown in Figure 5.

TktBookS Class

The TktBookS class has one instance variable: nextTicketNum. The only method that the TktBookS class supports is the getTicket method.

The most interesting classes in this model are Valet and Garage. These classes contain the behavior in the valet parking system to store and retrieve cars to and from a garage.

As you read subsequent articles in this issue, refer to this article for the definitions of the classes used in the valet parking system. You can find a complete copy of the SOM implementation of the valet parking system on the World-Wide Web at <http://pscc.dfw.ibm.com/psmag/>.

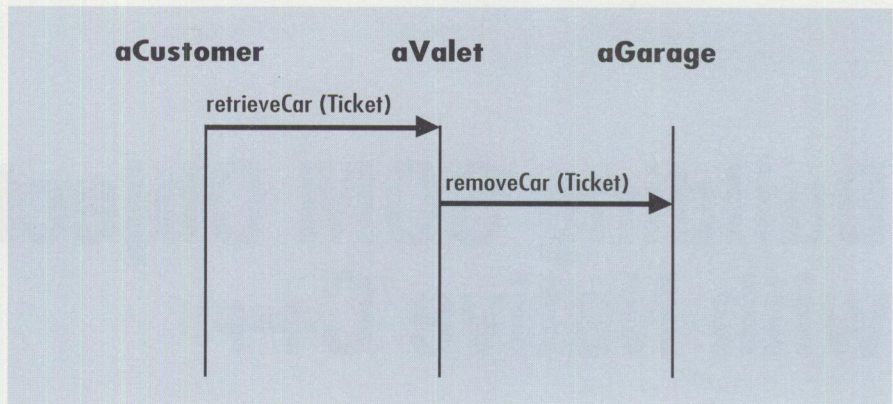


Figure 4. Object Interaction Diagram for retrieveCar Method

```

/* Garage.idl*/
#include <somobj.idl>
#include <Car.idl>

interface Ticket;
interface Garage : SOMObject
{
    exception carMissing
    {
        long errorCode;
        char reason[80];
    };

    const long PARKING_SPACES=1000;

    attribute sequence <Car, PARKING_SPACES> garage_lot;

    void storeCar(in Car aCar, in Ticket aTicket);
    Car removeCar(in Ticket aTicket) raises(carMissing);
    void listCars();
    long getCarCount();
    long getAvailableParkingSlot();
    void removeAllCars();

#ifdef __SOMIDL__
implementation
{
    passthru C_xh_before = "#include \"Ticket.xh\""
                          "#include \"Car.xh\""
                          "#include <iostream.h>";

    dllname = "Garage.dll";
    releaseorder : storeCar, removeCar, getCarCount, listCars,
                  getAvailableParkingSlot, removeAllCars;
    somDefaultInit: override, init;
    somDestruct: override;
};
#endif
};
  
```

Figure 5. Garage Class IDL (Garage.idl)



Rick Weaver is an advisory marketing support representative in the IBM Personal Systems Competency Center in Roanoke, Texas. He provides services and on-site consulting on DB2 for OS/2, DB2 for NT, SOM, and object technology to customers throughout the world. Rick joined IBM in 1990 after earning a BS degree in Computer Science from Southwestern Adventist College. His Internet ID is weaveros2@vnet.ibm.com.

Building SOM Objects with Native C++

You may think that using SOM with IDL (interface definition language) is great for new applications. But what about existing C++ applications? Will you need to rewrite your class definitions from C++ to IDL to take advantage of SOM and DSOM? With Direct-to-SOM (DTS), the answer is a reverberating NO! This article discusses how you can use Direct-to-SOM to build SOM objects from C++.

Editor's Note: *The code examples in this article refer to a simple object model discussed in Rick Weaver's article "IBM System Object Model—The Wave of the Future (and Now!)."*

Direct-to-SOM (DTS) provides an easy way to natively build ordinary C++ classes as SOM classes within C++. Normally, if you develop a SOM class, you create the class interface in IDL, then run the IDL through the SOM compiler (SC), and the SOM compiler generates a binding for the target language.

If the SOM compiler emits language bindings for C++ for the Valet class, you normally generate the following files:

- Valet.xh (interface header file)
- Valet.cpp (stub implementation source)
- Valet.xih (implementation header file)

Rick Weaver
IBM Corporation
Roanoke, Texas

If the Valet class is a new class, then using the SOM compiler is probably a good choice for developing the Valet class. But what if the Valet class was previously written in C++ (as Figure 1 shows)? You can manually rewrite your Valet class definition to IDL and convert Valet to a SOM class, but it's easier and more natural for C++ programmers to use a DTS compiler.

With a DTS compiler, you can build SOM classes directly in C++ without writing a single line of IDL.

Currently, the IBM VisualAge C++ and MetaWare High C++ compilers allow you to natively build DTS classes. The examples in this article use the IBM VisualAge C++ compiler.

Converting a C++ Class to a DTS Class

Assuming you have a class definition that can be supported under SOM,¹ you'll find it easy to convert your C++ class to a DTS class.

One way to convert a C++ class to a DTS class is to include the header file

¹ SOM and C++ are slightly different in their view of objects. For example, SOM does not support multiple virtual inheritance.



#include <som.hh> before the first definition of the class you want SOM-enabled. Once <som.hh> is included, the class you want SOM-enabled will need to be derived publicly from SOMObject. Figure 2 shows the Valet class defined as a SOM class.

One significant benefit of using SOM attributes is that they can automatically generate `_get` and `_set` member functions. If you want one of your data members treated as a SOM attribute, so that the `_get` and `_set` member functions are automatically generated, you must use the `#pragma SOMAttribute (member-data name,option)` within the class definition where the data member is defined. If you do not use `#pragma SOMAttribute`, the data members are considered to be instance variables, and no `_get` and `_set` member functions are generated. In Figure 2, if you want to make `valetName` an attribute, you must define the pragma within the Valet class: `#pragma SOMAttribute (valetName, privatedata)`.

When defining C++ classes with SOM attributes, keep in mind that the access level to the `_get` and `_set` member functions is the same as the data member's. If you define the `valetName` in Figure 2 to be a SOM attribute through the `#pragma`, the `_get` and `_set` member functions behave as if they are defined in the private section of the class definition. Member data defined as SOM attributes in the public section of a class will have `_get` and `_set` member functions defined in the public section.

If you want to define your data members as private member data, and you want to access this member data through public `_get` and `_set` members, you will need to define your own `_get` and `_set` member functions, place them in the public section of your class, leave your member data as SOM instance data, and not make it an attribute.

Another way to convert a C++ class to a SOM class is to use the `/Ga VisualAge C++` compiler option (enable implicit SOM mode). Using the `/Ga` option will implicitly include `som.hh` and implicitly derive classes from `SOMObject`. Using `#pragma SOMAsDefault(on)` in a class will achieve the same result as using the `/Ga` compiler option. If you do use `#pragma`

```
#ifndef __valetHH__
#define __valetHH__

#include <Garage.idl>
#include <TktBookS.idl>

class Car;

class Valet
{
public:
    Ticket* parkCar(Car *);
    Car* retrieveCar(Ticket *);
private:
    char    *valetName;
    long    valetID;
    Garage  *aGarage;
    TktBooks *aTktBookS;
};
#endif
```

Figure 1. Valet.hpp Written in C++

```
#ifndef __valetHH__
#define __valetHH__

#include <som.hh>

#include <garage.hh>
#include <TktBooks.hh>

class Car;

class Valet : public SOMObject
{
public:
    Ticket* parkCar(Car *);
    Car* retrieveCar(Ticket *);
private:
    char    *valetName;
    long    valetID;
    Garage  *aGarage;
    TktBooks *aTktBookS;
};
#endif
```

Figure 2. Valet.hh Defined as a SOM Class

`SOMAsDefault(on)`, all classes will be converted to SOM classes until `#pragma SOMAsDefault(off)` is encountered.

Building a Shared Library

If you are building a class library with your DTS class, you will need to export the four symbols (shown below) so that this class can be used by other clients. If you are going to place the Valet class in a shared library (DLL), the module definition file (.DEF) should contain the following:

```
EXPORTS
    ValetClassData
    ValetCClassData
```

```
ValetNewClass
SOMInitModule
```

The `SOMInitModule` is required for any shared library that can be dynamically loaded using methods supported by the SOM Class Manager. The `SOMInitModule` function for the Valet class is shown in Figure 3. The `SOMInitModule` can be included within the body of the class implementation (.CPP) or in a separate .CPP file, then linked together when the shared library is built.

```

void _Export _System SOMInitModule(long majorV,
                                   long minorV,
                                   char *className)
{
    ValetNewClass(Valet_MajorVersion, Valet_MinorVersion);
}

```

Figure 3. Initialization Function for Valet Class in SOM Class Library

```

...
class Valet : public SOMObject
{
public:
    long getCarCount();
    Ticket* park(Car *);
    Ticket* park(Bus *);
    Valet& operator=(const Valet& rhs);
    ...
};
#endif

```

Figure 4. Valet Overloaded Methods

```

...
class Valet : public SOMObject
{
public:
    #pragma SOMNoMangling(ON)
    long getCarCount();
    #pragma SOMNoMangling(OFF)
    Ticket* park(Car *);
    #pragma SOMMethodName(park(Car *),"parkCar")
    Ticket* park(Bus *);
    #pragma SOMMethodName(park(Bus *),"parkBus")
    Valet& operator=(const Valet& rhs);
    #pragma SOMMethodName(operator=(),"ValetEqualityTest")
    ...
};
#pragma SOMClassName(valet,"Valet")
#endif

```

Figure 5. Valet Overloaded Methods

SOM Release Order in a DTS Class

SOM offers two major benefits: binary compatibility and the ability to change methods and add new methods without forcing clients using a class library to recompile. The great news is that you can achieve this using a DTS class.

SOM maintains the binary compatibility through a release order list containing every method introduced by a class. Because a DTS object is a SOM object, there is a release order as well. You can specify the release order of a class through a `#pragma SOMReleaseOrder`.

For the Valet class in Figure 2, the release order is defined as: `#pragma SOMReleaseOrder(parkCar(),\ retrieveCar())` and the `#pragma SOMReleaseOrder` directive is specified within the class definition of Valet.

If any of the member data in the Valet class is defined as SOM attributes through the `#pragma SOMAttribute()` directive, the `_get` and `_set` member functions that are automatically generated for the attribute will also need to be included in the release order.

If you don't specify `#pragma SOMReleaseOrder`, VisualAge C++ will

generate a default release based upon the methods declaration order and SOM attributes in the class definition. Once a release order is created, you should not reorder the declaration of methods or attributes, because doing so changes the release order of the SOM class. Unless they recompile, any clients using the old release order will be unable to use the new SOM class with the new release order.

You can use the `/Fr` compile option to view the implicit release order for a class. VisualAge C++ will write the release order of the class to standard output. You can then cut and paste the release order to the `.hh` file and create a release order through the `#pragma SOMReleaseOrder` directive.

Building Language-Neutral DTS Classes

As many C++ developers know, the names of C++ functions and methods are mangled by the compiler. This also happens if you are building DTS classes in C++. If you generate IDL from your DTS class, by default, the mangled method will be imbedded in the IDL. If any client wants to use your DTS from any language outside of C++, the client will have to use the mangled name to invoke the desired method.

For example, say you want to add a few new methods to the Valet class. One method you want to add has a `long getCarCount()`; signature. After compiling Valet, the `getCarCount` method is mangled by VisualAge C++ to `getzcarzcount__fv`.

Any non-C++ class must refer to this mangled name when invoking the `getCarCount` method, making it difficult and error-prone to use a DTS class from a language other than C++. Luckily, with DTS under VisualAge C++, you can turn off mangling for a class and specify method names for methods that will need to be mangled!

Let's also add two overloaded functions to overload the `=` operator in Valet, as shown in Figure 4. Keep in mind that all functions, including overloaded functions and overloaded operators, will be mangled.

Overloaded Functions and Operators

SOM does not natively support overloaded functions and operators. To access the overloaded functions and methods in a language-neutral way through SOM (without using the mangled name), you can use `#pragma SOMMethodName` to give the DTS method name a SOM name. For the park methods in Figure 4, `#pragma SOMMethodName` is defined as:

```
#pragma SOMMethodName
    (park(Car *), "parkCar")
#pragma SOMMethodName
    (park(Bus *), "parkBus")
```

Any non-C++ client using this Valet class could now invoke the `park(car *)` method via the name `parkCar(Car *)` method and `park(bus *)` via the name `parkBus(Bus *)` method.

The overloaded operator (`operator=`) can also be renamed using the `#pragma SOMMethodName` directive. Any non-C++ client invokes the `operator=` method through the SOM method name specified in the `#pragma`.

As described, all methods, including overloaded member functions, will be mangled. In Figure 4, the `getCarCount` method will be mangled to `getzcarzcount__fv`. If you want all non-overloaded methods within a DTS class to have the same SOM name as the method name, you can use the `#pragma SOMNoMangling` directive.

Figure 5 shows the Valet class with the appropriate `#pragma` directives defined to ensure that the methods are easy to invoke from other languages. For any overloaded member functions, you must use the `#pragma SOMMethodName` with the `#pragma SOMNoMangling`, because every SOM method must have a unique name.

Additionally, C++ may also mangle the class name of a DTS class, which makes it more difficult to use from non-C++ languages. In the case of the Valet class, the mangled class name is `zvalet`. However, you can either specify the `#pragma`

`SOMNoMangling` directive before the class definition so the SOM class name is not mangled, or use `#pragma SOMClassName` (as illustrated in Figure 5) to specify a non-mangled class name.

Watch Those Constructors!

When building your DTS classes, your class will always need to provide a default constructor that takes no arguments. This SOM requirement is slightly different than in C++. If a client application tries to instantiate an object using a non-existence default constructor, a compile error will occur. If a default constructor is specified in the class, the IBM VisualAge C++ compiler will generate the following function:

```
somDefaultInit(this, Environment
    *, InitVector *);
```

This function will be invoked when an object is instantiated with a default constructor. If your DTS class contains other non-default constructors, the constructor names will be mangled like any other C++ member function. You should use the `#pragma SOMMethodName` directive to specify a SOM method name for all non-default constructors. If you want these non-default constructors to be used by non-C++ languages, these languages must first instantiate the SOM object through `SOMNewNoInit`. Once created, the client can then invoke the non-default constructor through the SOM name specified in the `#pragma SOMMethodName` directive.

If your DTS client class is C++, then the `#pragma SOMMethodName` directives are not necessary for the constructors.

DTS Classes with DSOM

Since a DSOM object exists in a separate process from the client application, no direct access to member data is allowed. Any member data defined as a SOM attribute must be accessed through `_get` and `_set` methods. Non-attribute member data must be accessed through `_get` and `_set` methods provided by the class. Of course, all good object programmers never allow direct access to member data, so this should not be a consideration.

If your DTS class is going to be built into a DLL, you should also ensure that the DLL name is placed in the interface repository (IR) so that the DLL can be loaded in DSOM. To accomplish this, you must use `#pragma SOMIDLPass` so that you can imbed the DLL name in the IDL. The `#pragma SOMIDLPass` directive will imbed text specified in `#pragma` in the IDL.

If the Valet class will be compiled into a DLL named Valet, imbed `#pragma SOMIDLPass` in the Valet class definition as follows:

```
#pragma SOMIDLPass
    (*, "Implementation-Begin",
    "dllname = \"valet.dll\";")
```

When you build the interface definition language for Valet from `Valet.hh`, `Valet.idl` will have the `dllname="valet.dll"` imbedded in the implementation section of the IDL. You can then run `Valet.IDL` through the SOM compiler to add the Valet class definition to the interface repository. DSOM will load the DLL and invoke methods dynamically.

SOM Objects from C++

Direct-to-SOM lets you easily build SOM objects from C++ without writing a single line of IDL. It allows C++ programmers to develop objects in a natural manner. Although C++ and SOM view objects slightly differently, with the `#pragma` directives provided with IBM VisualAge C++ and especially the `#pragma SOMMethodName`, you can develop your methods to be used by any language.



Rick Weaver is an advisory marketing support representative in the IBM Personal Systems Competency Center in Roanoke, Texas. He provides services and on-site consulting on

DB2 for OS/2, DB2 for NT, SOM, and object technology to customers throughout the world. Rick joined IBM in 1990 after earning a BS degree in Computer Science from Southwestern Adventist College. His Internet ID is weaveros2@vnet.ibm.com.

Distributing Objects with DSOM

Using the same valet parking system scenario described in Rick Weaver's "IBM System Object Model—The Wave of the Future (and Now!)" article, this article provides you with background information for distributing objects with Distributed SOM (DSOM) and walks you through the necessary changes to convert the valet sample into a DSOM application.

DSOM is an extension to SOM that allows client applications to access objects without building dependencies upon the object's location into the client application's source code. It also allows clients to invoke methods on SOM objects in other processes.

By providing a framework for accessing objects across address spaces within a machine or between machines, DSOM insulates clients from each object's location. DSOM objects can be similarly referenced, regardless of their locations.

Randall Autry
IBM Corporation
Roanoke, Texas

Usage

DSOM is intended for applications that need to share objects within multiple programs. The object exists in only one process, called the server, and clients access the object using remote method invocations made transparent by DSOM. DSOM should be used for objects that need to be isolated from the main program for reliability, portability, or distributed design.

Features

The following is a quick summary of important DSOM features. DSOM:

- Uses the standard SOM compiler, interface repository (IR), language bindings, and class libraries, plus DSOM provides a growth path for non-distributed SOM applications.
- Allows an application program to transparently access a mix of local and remote objects.
- Provides runtime services for creating, destroying, identifying, locating, and



dispatching methods on remote objects. You can override or augment these services to suit the application.

- Uses existing interprocess communication (IPC) facilities for workstation communication and common LAN transport facilities for workgroup communications.
- Provides support for writing multi-threaded servers and event-driven programs.
- Provides a default object server program that can be easily used to create SOM objects and make those objects accessible to one or more client programs. If the default server program is used, SOM class libraries are loaded upon demand, so no server programming or compiling is necessary.
- Complies (as an object request broker [ORB]), with the Common Object Request Broker Architecture (CORBA) specification published by the Object Management Group (OMG) and X/Open, which is important for porting applications to other CORBA-compliant ORBs.

DSOM Overview

A DSOM application typically consists of four processes running on a single machine or across multiple machines:

- The *client program* uses DSOM classes and is fully custom code written by the application developer.
- The *server program* may be the default server program provided by DSOM or a customized server program written by the application developer. The default server program simply runs in a loop, listening for and servicing client requests. It hosts a well-known server object, which responds to generic methods for loading and instantiating application-specific class libraries, and it hosts the application objects created in it. For applications requiring more flexibility or functionality, a customized server program addresses the application's specific needs.
- The DSOM *location-service daemon*, SOMDD, runs on the same machine as the servers. The daemon establishes the initial connection between client and server and dynamically starts the server program on the client's behalf, if necessary.

```
#include <somobj.idl>
#include <garage.idl>
#include <TktbookS.idl>

interface Car;

interface Valet : SOMObject
{
  attribute string valetName;
  attribute long valetID;

  Ticket parkCar(in Car aCar);          /*parkCar method*/
  Car retrieveCar(in Ticket aTicket);    /*retrieveCar method*/
}

#ifdef __SOMIDL__
implementation
{
  Garage aGarage;
  TktBookS aTicketBookSystem;

  passthru C_xh =
    "#include \"Ticket.xh\""
    "#include \"Car.xh\""
    "#include \"Garage.xh\""
    "#include <TktBooks.xh>"
    "#include <string.h>";
    //Previous line added for DSOM

  dllname = "valet.dll";
  releaseorder : _get_valetName, _set_valetName, _get_valetID,
                _set_valetID, parkCar, retrieveCar;

  //Next line added for DSOM - must perform "deep copy", not
  // "shallow copy"
  valetName : noset;

  somDefaultInit : override, init;
  somDestruct : override;
};
#endif
};
```

Figure 1. VALET.IDL

- The *name server* provides a Naming Service used directly by DSOM applications. DSOM also uses the name server to provide a Factory Service used by client programs to create remote objects.

The DSOM application uses the following files at runtime:

- The *SOMobjects configuration* file defines runtime environment settings for DSOM. Each of the four DSOM processes just described can have unique configuration file settings, or they can share a common configuration file. SOMobjects provide a default configuration file, which you can customize using any text editor.
- The *interface repository (IR) files* primarily load class libraries dynamically in both client and server processes. IR files are created and updated using the SOM compiler and contain information required only on the server the daemon

uses to start servers and required by servers to initialize themselves. This repository is created and updated by using `regimpl` or `pregimpl` to register servers.

- *Naming Service files* store information from the Naming Service and the DSOM Factory Service on disk. Naming Service files include information about which application classes are supported on each registered server.

You'll perform the following tasks to configure and run a DSOM application:

- Customize the environment settings by editing the default configuration file and using the `somenv` environment variable.
- Configure the Naming Service and Security Service using the `som_cfg` utility (a one-time step). Update the interface repository to include application interface definition language (IDL).

Note: Some code is omitted here to save space.

```
/*
 *Method from the IDL attribute statement:
 **attribute string valetName"
 */
SOM_Scope void SOMLINK _set_valetName(Valet *somSelf, Environment
                                     *ev, string valetName)
{
    ValetData *somThis = ValetGetData(somSelf);
    ValetMethodDebug("Valet", "_set_valetName");

    //Added for DSOM since this method is only stubbed because of
    "noset"
    if (somThis->valetName != 0) {
        SOMFree(somThis->valetName);
    } /* endif */

    somThis->valetName = (char *) SOMMalloc (strlen(valetName) + 1);
    strcpy(somThis->valetName, valetName);
}
```

Note: Some code is omitted here to save space.

```
/*
 * SOM_Scope void SOMLINK somDestruct(Valet *somSelf, octet doFree,
                                     somDestructCtrl* ctrl)
 */
/*
 * The prototype for somDestruct was replaced by the following
 * prototype:
 */
SOM_Scope void SOMLINK somDestruct(Valet *somSelf, octet doFree,
                                   som3DestructCtrl* ctrl)
{
    ValetData *somThis; /* set in BeginDestructor */
    somDestructCtrl globalCtrl;
    somBooleanVector myMask;
    ValetMethodDebug("Valet", "somDestruct");
    Valet_BeginDestructor;

    /*
     * local Valet deinitialization code added by programmer
     */
    delete somThis->aGarage;
    delete somThis->aTicketBookSystem;

    //Following lines added for DSOM
    if (somThis->valetName!=0) {
        SOMFree(somThis->valetName);
    } /* endif */

    Valet_EndDestructor;
}
```

Figure 2. VALET.CPP

- Start SOMDD on the server machines.
- Register application servers and classes using the `regimpl` or `pregimpl` tool.
- Run the client application.

At runtime, DSOM clients and servers communicate through proxy objects (object references). A *proxy object* locally represents a remote target object. A proxy inherits the target object's interface, so it responds to the same methods. Operations invoked on the proxy do not execute locally, but are forwarded to the real target object for execution. The client

program treats a proxy object much like a local object and always has a proxy for each remote target object on which it operates. The proxy forwards requests and yields results from the remote object.

Changing Valet to Use Distributed Objects

The following steps outline the changes necessary to the valet sample to distribute several of the objects in the valet sample. This sample is based on the SOMObjects 3.0 Beta Toolkit instead of the SOMObjects 2.1 Toolkit used in the original sample. The differences in code are minor and

will be pointed out separately from the changes for distributing Valet. You can find the complete code used for this sample on *Personal Systems'* home page at <http://pscc.dfw.ibm.com/psmag/>.

Although SOMObjects 3.0 supports several methods for distributing the valet example, including using factory objects for better location transparency, I have outlined a simple solution here. Valet, Car, and Ticket will be converted to DSOM objects to allow the valet system to be distributed across process or machine boundaries.

The only change required to the valet sample for use with SOMObjects 3.0 Beta Toolkit instead of SOMObjects 2.1 Toolkit is to alter *.MAK files to use `SOMTK1.LIB` and `SOMTK2.LIB` in place of `SOMTK.LIB` to use with `ILINK`.

Following are the programming changes required for the valet sample to use DSOM objects:

Distributing Valet

Although valet is partly ready for distributing, you must change the `VALET.IDL`, `VALET.CPP`, and `CLIENT.CPP` files.

In `VALET.IDL` (see Figure 1):

- Use `noset` to prevent the emitter from creating a `_set` method for `valetName`. You must customize the `_set` method for `valetName` for DSOM to perform a "deep" copy of the string passed in, since the pointer will not be valid in a separate address space. Add `valetName : noset;`

In `VALET.CPP` (see Figure 2):

- Implement a deep copy for `_set_valetName()` that will work in a separate address space where a direct pointer assignment would be invalid. Add the following code to `_set_valetName()`:

```
if (somThis->valetName != 0) {
    SOMFree(somThis->valetName);
} /* endif */

somThis->valetName = (char *)
    SOMMalloc (strlen(valetName)+1);
strcpy(somThis->valetName,
       valetName);
```

Note: Comments are omitted.

```
#include "valet.xh"
#include "car.xh"

//Following line added for DSOM
#include <some.xh>

#include <fstream.h>
#include <string.h>
#include <stdlib.h>

long check_ev(Environment *); //Used to check SOM exceptions

main(int argc, char *argv[], char *envp[])
{
    fstream infile;
    char buf[80]; /*buffer for holding record from file*/
    char *token=0; /*used when reading input file*/
    long tNum=0;

    Environment *ev = somGetGlobalEnvironment();

    //Following lines added for DSOM
    SOMD_Init(ev);
    ValetNewClass (0,0);
    CarNewClass (0,0);
    TicketNewClass (0,0);

    //Following change made for DSOM
    // Valet *v = new Valet();
    Valet *v = (Valet *) SOMD_ObjectMgr->somdNewObject(ev, "Valet", "");
    Ticket *t;

    /*Create a valet object*/
    v->_set_valetName(ev,"Roy Bigg");
    v->_set_valetID(ev,1);

    infile.open("car.scr",ios::in); // *** open car.scr for input
    while (!infile.eof())
    {
        buf[0]='\0';
        infile.getline(buf,sizeof(buf),'\n');
        if (strlen(buf)!=0) // *** if it is a valid record
        {
            //Following change made for DSOM
            // Car *c = new Car();
            Car *c = (Car *) SOMD_ObjectMgr->somdNewObject(ev, "Car", "");

            token=strtok(buf,"|");
            if (*token=='P') //Park a car.
            {
                //Moved for consistency
                Car *c = (Car *) SOMD_ObjectMgr->somdNewObject(ev, "Car", "");

                token=strtok(NULL,"|");
                c->_set_color(ev,token);
                token = strtok(NULL,"|");
                c->_set_make(ev,token);
                token = strtok(NULL,"|");
                c->_set_model(ev,token);
                t = v->parkCar(ev,c);
                cout << "=> The "<<c->_get_color(ev)<<" "<<c->_get_make(ev)<<"
                    "<<c->_get_model(ev)<<" was parked in slot ";
                cout <<t->_get_parkingSlotNum(ev)<<endl;

                delete t;
            }
            else //Retrieve a car.
            {
                //Following change made for DSOM
                //Ticket *nT = new Ticket;
                Ticket *nT = (Ticket *) SOMD_ObjectMgr->somdNewObject(ev, "Ticket", "");

                token=strtok(NULL,"|");
                tNum=atol(token);
                nT->_set_parkingSlotNum(ev,tNum);
                c = v->retrieveCar(ev,nT);
                if (!check_ev(ev))
                    cout << "=> The "<<c->_get_color(ev) << " " << c->_get_make(ev) << "
                        "<<c->_get_model(ev)<<" was returned from slot "<< nT->
                            _get_parkingSlotNum(ev)<<endl;
            }
        }
    }
}
```

Figure 3. CLIENT.CPP (continued on next page)

```

//Following changes made for DSOM
//delete nT;
//delete c; //delete car object returned.
    SOMFree (nT);
    SOMFree (c);
    }/*else*/
    }/*if..then*/
    }/*while*/
infile.close();

//Following change made for DSOM
// delete v;
SOMFree (v);
SOMD_Uninit(ev);
}

(Function check_ev(Environment *ev) omitted)

```

Figure 3. CLIENT.CPP (continued from previous page)

```

#include <somobj.idl>
interface Ticket : SOMObject
{
    attribute long ticketNum;
    attribute long parkingSlotNum;
#ifdef __SOMIDL__
    implementation
    {
        dllname = "ticket.dll";
        releaseorder : _get_ticketNum, _set_ticketNum_get_parkingSlotNum,
            _set_parkingSlotNum;

        //Following two lines added for DSOM
        somDefaultInit : override, init;
        somDestruct : override;
    };
#endif
};

```

Figure 4. TICKET.IDL

```

Implementation id.....: 30eale82-001520ae-7f-00-000000002079
Implementation alias.....: ValetServer
ImplDef Class name.....: ImplementationDef
Program name.....: somdsvr
Multithreaded.....: No
Server managed.....: No
Server secure.....: No
Server class.....: SOMDServer
Configuration file.....:
Protocol information.....:
    Protocol: SOMD_IPC;Hostname: thehostname;Port: 3002;

```

Figure 5. Implementation for ValetServer Alias

- To properly free the memory allocated for valetName, add the following code to somDestruct():

```

if (somThis->valetName!=0) {
    SOMFree(somThis->valetName);
} /* endif */

```

In CLIENT.CPP (see Figure 3):

- Include DSOM definitions by adding `#include <somd.xh>`
- Use `SOMDNewObject` to instantiate DSOM objects, delete new Valet; and add:

```

(Valet *) SOMD_ObjectMgr-
    >somdNewObject(ev, "Valet", "");

```

- Use `SOMFree(v)` instead of `delete v` to free DSOM objects
- Add `SOMD_Init(ev)` to initialize the DSOM environment
- Add a dummy class object: `ValetNewClass (0,0)`
- Add `SOMD_Uninit(ev)` to de-initialize DSOM

Distributing Car

The Car object is already DSOM-ready, so you need to change only the CLIENT.CPP file.

In CLIENT.CPP (see Figure 3):

- Add `CarNewClass (0,0);`
- Remove `Car *c = new Car();`
- Add `Car *c = (Car *) SOMD_ObjectMgr->somdNewObject (ev, "Car", "");`
- Remove `delete c;`
- Add `SOMFree (c);`

Distributing Ticket

Because of its initial implementation, Ticket requires changes to CLIENT.CPP and TICKET.IDL.

In CLIENT.CPP (see Figure 3):

- Add `TicketNewClass (0,0);`
- Remove `Ticket *nT = new Ticket;`
- Add `Ticket *nT = (Ticket *) SOMD_ObjectMgr>somdNewObject (ev, "Ticket", "");`
- Remove `delete nT;`
- Add `SOMFree (nT);`

In TICKET.IDL (see Figure 4):

- Add `somDefaultInit : override, init;\`
- Add `somDestruct : override;\`

Configuring DSOM Applications

You must configure the DSOM runtime environment before DSOM applications will run. To do this, perform the following configuration tasks:

- Set environment variables and configuration file settings. DSOM settings

are stored in the SOMobjects configuration file in the [somd] stanza. Most DSOM settings are optional; the default settings provided by the default SOMobjects configuration file are suitable for single-machine applications. Set the SOMENV environment variable for accessing the configuration file.

- If you are using the SOMobjects Security Service, users must log in to the LAN server to run as authenticated clients. Users not logged in to the LAN server will run as unauthenticated clients; their requests will be rejected by any server registered as a secure server. If you are not using the SOMobjects Security Service, set `DISABLE_AUTHN=TRUE` in the [somsec] stanza of the SOMobjects configuration file.
- Use the SOM Compiler (with the `ir` emitter) if any application classes need to be compiled into the interface repository.
- Ensure that the Naming Service has been previously configured with the `som_cfg` tool. Even though this is typically a one-time configuration step, you should reconfigure the Naming Service as necessary when certain DSOM communication-protocol environment settings are changed.
- Use the `regimpl` or `pregimpl` tool to register all application servers. In addition, register all application class/server associations needed by the DSOM Factory Service.
- Use the `somdchk` tool to verify the DSOM runtime environment.

Configuring DSOM for Valet

To configure DSOM for the valet sample:

- Run `somdchk` and fix any problems identified
- Set `beginlibpath=x:\valet\dll;`
- Set `somir=%somir%;x:\valet\valet.ir;`
- Set `somddir=x:\valet\somddir\`
- Run `regimpl` and add an implementation alias of ValetServer with all defaults
- Use `regimpl` to add Valet, Car, and Ticket classes, specifying ValetServer as the implementation alias for each

When you use selection "4. Show one" to display the ValetServer alias, it should look like Figure 5.

Running DSOM Applications

To run a DSOM application, you must start SOMDD on each server. For the valet sample, you can start SOMDD from the command line by typing `START SOMDD`. After SOMDD is up and running, you can start server applications manually or automatically when requested by clients. The client program is run on either the same or a different machine from the server and will connect automatically with SOMDD, locating the appropriate server based on the configuration.

Before running the distributed version of valet, edit `CAR.SCR` to remove the invalid references used to show exceptions. (Since exceptions were not covered here, the

code produced will not handle them correctly.) Remove line 10 `R|100` and line 31 `R|30`.

Ensure that the environment variables you set earlier are set for your current OS/2 session. Start SOMDD, wait for the Ready message, and start CLIENT. You should see the same list of cars produced in the previous examples, minus the exceptions produced in the script.

References

- *SOMobjects Developer Toolkit Programmer's Guide, Volume 1: SOM and DSOM*
- *SOMobjects Developer Toolkit Programmer's Reference*

Note: These references are included on the CD-ROM with the software.



Randall Autry is an OS/2 application development and communication specialist on the advanced object technology team in the IBM Personal Systems Competency Center,

Roanoke, Texas. He designs and implements software systems, specializing in OS/2 and object technology and has focused for five years on client/server application design and implementation, problem determination, and consulting. Randall's Internet ID is `autry@vnet.ibm.com`.

Using OpenDoc and SOM in Application Development

The initial version of OpenDoc is now available for OS/2. This article explains what OpenDoc is and reviews the content of a simple "Hello World" component.

Reacting to the growing interest for using compound document architectures to build applications, the initial version of OpenDoc has recently been made available for OS/2.

The concept behind compound document architecture is that any application can consist of modular pieces, each performing a specific task. There may be components for editing text, viewing images, playing back video and speech, or recognizing handwriting. These components can be developed and marketed by experts with specific domain expertise, then selected and assembled by either corporate professionals or by end users themselves.

This component-based application has many potential benefits. As monolithic applications add function, they grow larger and more difficult to maintain. The level of difficulty doesn't increase linearly—it follows a more exponential curve. Component architecture promises to help contain an application's size as well as reduce the difficulties developers experience when designing, coding, and testing, because each component is smaller and focused on more specific functionality.

John Pape
IBM Corporation
Roanoke, Texas

When components become widely available, today's requirements for a typical application such as word-processing with graphics and image capabilities, spelling and grammar

checkers, thesauruses, etc. will be met by interchangeable, off-the-shelf components provided by numerous vendors. In addition, users can add new functionality, such as adding World-Wide Web hyperlinks into a document to be electronically sent, by embedding a new component type into the document. Then the document can be viewed by the recipient using low- or no-cost component viewers that vendors are being encouraged to make available. These inexpensive component viewers will be easy to add to user workstations, adding function while eliminating the need to upgrade a large integrated package.

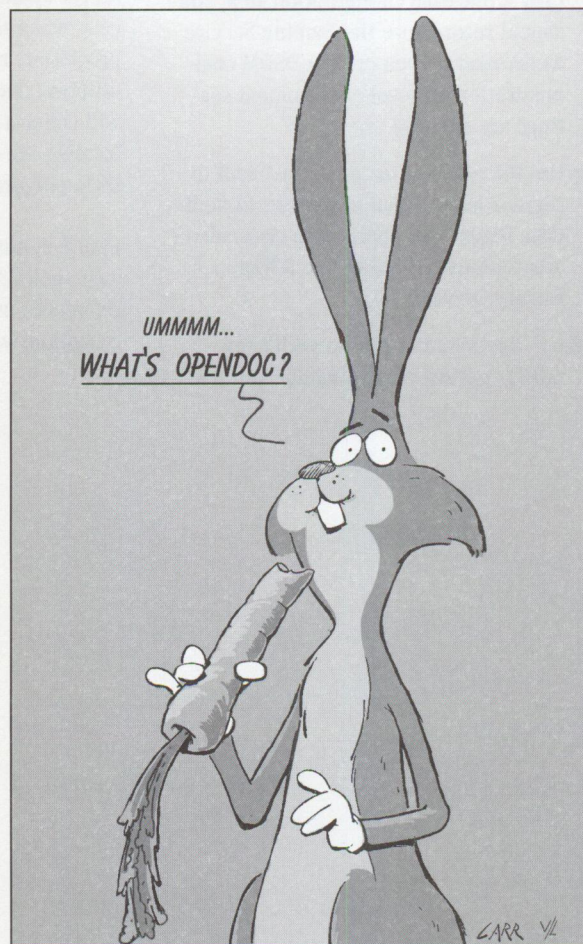
Several factors contribute to components' flexibility and power. Components conform rigorously to standards administered by an independent organization. Components add functionality,

such as scripting, linking, and embedding, that allows them to interoperate within a larger entity. Finally, components are typically packaged as relatively small dynamic link libraries (DLLs) that are easy to add to a workstation.

OpenDoc and SOM

How do OpenDoc and SOM fit into this application development vision?

OpenDoc is an open specification administered by Component Integration Laboratories (CILabs), an independent



organization whose members include many software vendors (originally Apple, IBM, Novell, Oracle, Sun, Taligent, WordPerfect, and Xerox; and later, Adobe, Lotus, MetaWare, and OMG). CILabs is responsible for arbitrating the specification of the standard for, and ensuring conformance of, developed components. CILabs also recognizes the presence of competing architectures in the marketplace, such as Microsoft's Object Linking and Embedding (OLE), and specifies an architecture known as Open Linking and Embedding of Objects (OLEO) to ensure a level of coexistence.

OpenDoc consists of several major facilities, as shown in Figure 1:

- Compound Document Management
- Structured Storage
- Automation and Scripting
- Uniform Data Transfer
- Object Bus

I'll discuss each of these facilities now.

Compound Document Management

Elements of OpenDoc's visual programming differ from that of a typical Presentation Manager (PM) or Windows application. In OpenDoc, the goal is to have multiple components seamlessly share a presentation space. Components must share access to various resources, such as the keyboard or a menu. The end user must be able to navigate within the document in a comfortable, natural way with minimum effort.

To address these requirements, OpenDoc provides an architecture in which components negotiate for visual space and invoke the services of an arbitrator to gain access to shared resources.

Usually, the space negotiation begins when the component is first embedded in a document. The definition of a *document* is rather broad; a document does not have to be thought of as word-processing output—it can be a full business application with communication links and database access. Today, the basis for all documents is the DocShell, which provides the runtime environment. The DocShell is shown with an embedded sample component in Figure 2.

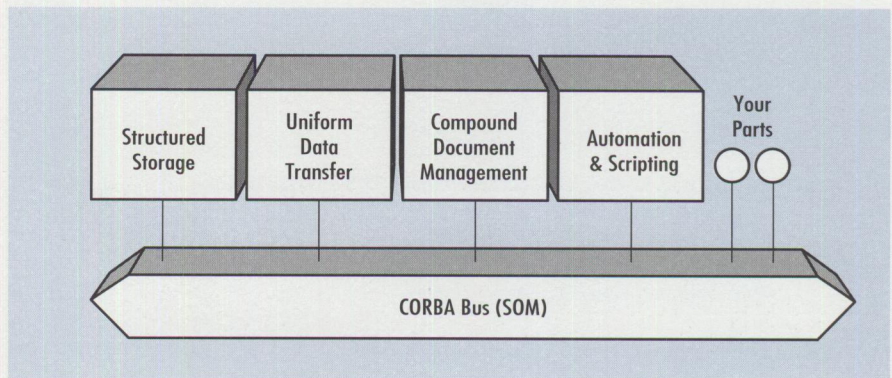


Figure 1. OpenDoc: The SOM Object Bus

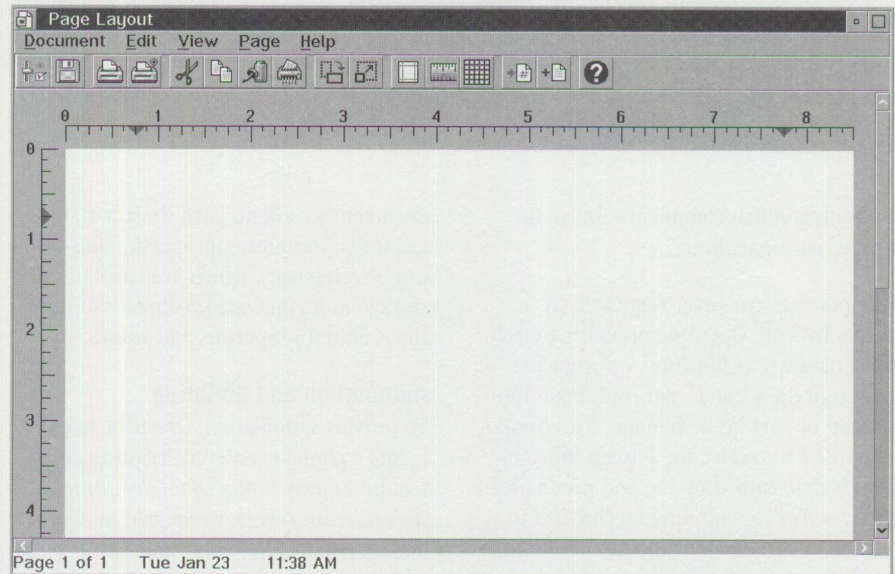


Figure 2. An OpenDoc Application

Components that allow other components to be embedded within them are known as *containers*. Space is allocated with frames, facets, and shapes. Negotiation for space takes place between components and their containers at the *frame* level. *Facets* are areas where the frame is visible; the presence of facets aids OpenDoc's performance. *Shapes* allow irregularly shaped components, such as text that wraps around images or a round clock.

OpenDoc differs from existing applications in the manner in which components are selected and manipulated. OpenDoc utilizes *inside-out activation*, which means that mouse clicks are directed to the event handler of the most deeply embedded facet containing the point clicked on. Activation, however, is not automatic. In handling the mouse event, the component can choose to issue a request for a specific focus. Further, the

focus is obtained only after negotiation, through the arbitrator, with the component currently holding the requested focus. Typically, the focuses that a component will request include access to keystrokes, ownership of the menu bar, and the selection highlight.

The inside-out activation scheme allows an aspect of the user interface known as *in-place editing*. This interface enables users to use or manipulate each of the embedded components without having to launch a separate application. Users can naturally navigate through a document by clicking on its various visual features.

A document received from another user contains information regarding embedded frames and the type of information content they represent, called their *part kind*. As shown in Figure 3, the user can configure a workstation with preferences

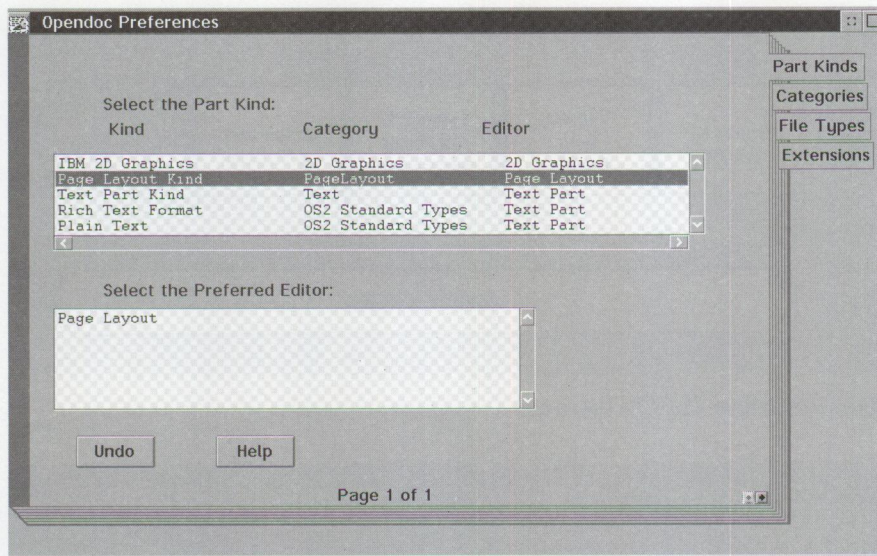


Figure 3. OpenDoc Preferences

indicating which component to use to render each part kind.

Components are never requested by name. Instead, OpenDoc provides a mechanism known as *binding*. A document may contain a frame that contains information of part kind "bitmap." The component used to render the bitmap information is determined by the end user's preferences. For this binding mechanism to work, vendors must cooperate on uniform methods of storing data content.

Structured Storage

OpenDoc provides a uniform method of storage and representation through a structured storage mechanism known as Bento, a mature technology used in more than 100 products. OpenDoc's compound document architecture exists largely because of Bento's capability for storing a large variety of different data types, such as text, video, and sound.

A Bento container consists of a self-contained indexing scheme and elements known as *storage units*, which contain properties and values. These properties and values are represented as ISO strings, a standardized scheme that allows data to be identified by a commonly known character sequence. This is the scheme that allows the binding mechanism described earlier.

Each physical file represents a document. Each document contains one or more drafts, which (if appropriate) allow the user to return to a previous state of the

document's content. Each draft has storage units, containing properties and values or references, which are links to other storage units that are traversed to locate the required properties and values.

Automation and Scripting

To provide automation, OpenDoc implements a content-centered scripting architecture known as the *Open Scripting Architecture (OSA)*, developed by Apple and modeled closely on AppleScript. This architecture provides a language-neutral way to specify *semantic events*, which are defined as scripting messages that manipulate the component's content model. The content model lists the content objects and the operations available to the scripting support.

By coding all routines that change a component's content model as semantic events, several advantages are gained. Most simply, the user can automate the interaction with the component, including doing everything that could have been done with the keyboard and mouse, by writing scripts instead. Second, interactions become recordable, so that any sequence that the user performs frequently, such as accessing data in a database through a complex series of queries, can be captured and replayed. Scripts also play a part in providing the "glue" for the component assembly process.

In the future, components may become intelligent enough to learn by example,

then automate entire workflows. Network-aware components could roam the Internet to perform a customized query on a number of servers.

Uniform Data Transfer

Uniform data transfer is accomplished through the use of three services: linking, drag and drop, and the clipboard. Components inherit a set of interfaces from the base class ODPart that support the event notification necessary to support these services. In addition, a set of classes is provided to initiate the services and manipulate the data being passed. These services are uniform in that they all use Bento storage units as the object-based data carrier, and each mechanism will accept the storage unit as a component to be embedded or as content data.

OpenDoc's linking support is an extension of Apple's publish and subscribe model. This scheme enables components (publishers) to offer source data to other components (subscribers) that want to stay synchronized. When establishing the link, the user can choose to have source updates received automatically or by request. Each time that the link source updates the source data and marks it as changed, the updated storage unit is transferred to the link destination, which will embed the data as a new component or add it to its own data content. The underlying Distributed SOM (DSOM) support allows linking to components within the same document, between documents (interprocess), or between systems (intersystem).

Drag and drop refers to the user's ability to select or "drag" a component or data container and "drop" it onto another object. OS/2 users are familiar with this interaction with the Workplace Shell. The source component packages data in response to a start drag event, and, if a drop occurs, the destination component is given the opportunity to accept the passed storage unit and decide how to incorporate it into its own content. The accepted object may either be added to the component's own data content, or, if the receiving component is a container and the storage unit does not represent its own part kind, it may be embedded as a component.

The clipboard allows an object to be transferred through an intermediate

location. The object, still represented by a Bento storage unit, is copied or moved (cut) to the clipboard and can be retrieved (pasted) at a later time by another component.

Object Bus

OpenDoc uses SOM as the underlying client/server middleware and requires that every machine it runs on have a CORBA-compliant SOM object request broker (ORB). To fully enable the exchange of components across an enterprise, OpenDoc will require the widespread adoption of Distributed SOM.

SOM adds several capabilities to the OpenDoc implementation:

- Components can be packaged as binary libraries and shipped as DLLs.
- Components can be extended through implementation inheritance.
- Components are dynamically bound and can be added to documents at any time.
- SOM provides transparent interprocess, intersystem communications.

Anatomy of an OpenDoc Application

As shown in Figure 4, coding an OpenDoc component essentially means providing behaviors for more than 60 SOM methods. The term *part* is widely used and is interchangeable with *component*. (Though “component” is the official term, many OpenDoc classes and methods use “part” in their names.)

An OpenDoc component has no “main” section—it is developed as a DLL. These methods are organized into 12 major categories: imaging, user interface events, part activation, binding, frames, facets, containing parts, embedding, drag and drop, linking, undo, and constructors/destructors. These categories are described next.

■ **Imaging**—The drawing code available to components is platform-specific, with OS/2 using PM calls to the graphics programming interface (GPI) or to WIN APIs. OpenDoc notifies your component that it needs to update its presentation space by invoking ODPart’s imaging methods. ODPart’s Draw method is similar to the WM_PAINT message received by PM applications.

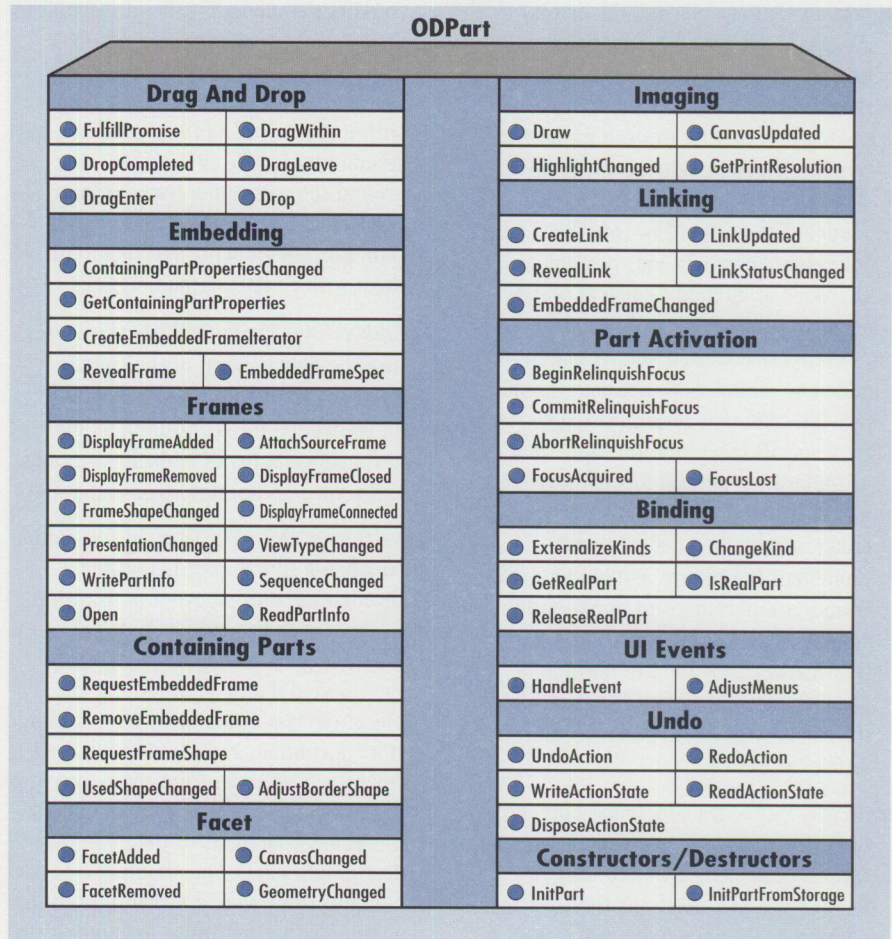


Figure 4. Methods of ODPart

- **User interface events**—These consist of two methods: HandleEvent and AdjustMenus. HandleEvent is similar to a PM application’s window procedure, except, through the help of ODDispatcher, the events received will correspond to the specific focus types held. AdjustMenus is used to dynamically alter the runtime state of the menu selections, such as graying out currently inactive choices.
- **Part activation**—Three of these methods (BeginRelinquishFocus, CommitRelinquishFocus, and AbortRelinquishFocus) are concerned with managing the transfer of focus away from your component. BeginRelinquishFocus allows you to declare that your component will relinquish the specific focus requested on this call. A commit scheme is used for focus sets; if all current focus holders have agreed to relinquish focus, then CommitRelinquishFocus allows any additional processing that your

component will require to record the commitment; otherwise, ODArbitrator will invoke AbortRelinquishFocus to notify your component that it still retains the focus. FocusAcquired and FocusLost allow your component to respond to unusual circumstances, such as losing a focus due to a system failure.

- **Binding**—A component’s metaclass provides the information necessary to update the part registration database as to the part kinds (data formats) supported. During runtime, a component only needs to externalize its preferred representation. ChangeKind can request that your component change its representation, such as changing the part kind of a graphics image from GIF to BMP. ExternalizeKinds will request that your component write several different representations to its storage unit based on a provided list.

■ **Frames**—Frames manage the display area available to your part. Frames encapsulate certain platform-specific structures, such as the `internal transform` that locates the frame relative to the component's content area and the `used shape` that defines the available area. OpenDoc interfaces with your component's frames through several groups of methods, some of which allow you to provide code in response to events that caused your frame to be added or removed. Your frame should provide code that responds to a request for a specific presentation type, such as a tabular or graphical format. A request for a particular view type will request either a frame (normal window), thumbnail (miniature window), or icon. Frames are permitted to persistently store data about themselves in a part information area and are notified of when to internalize and externalize their data with `ReadPartInfo` and `WritePartInfo`.

■ **Facets**—Facets are concerned only with the imaging aspects of the component. Facets encapsulate an `external transform` that positions the facet in relation to the containing frame. Facets contain a `clip shape` that allows a facet to create an irregular shape; drawing will always clip to this shape, so as to not interfere with the drawing of other facets. Facets also contain an `active shape` that defines the hit region for mouse events; this is used by the dispatcher to route mouse-related messages. `ODPart`'s facet-related methods should respond to the addition or removal of a facet. In a component where the content area is larger than the screen's display area, facets may be dynamically added and removed as they are scrolled in and out of view. Facets are temporary—they exist as a set of structures parallel to the frame, and they exist to enhance drawing and hit testing performance.

■ **Containing parts**—Frames are managed by containers. If a contained component requires a new display frame, the component will invoke the container's `RequestEmbeddedFrame` method. Negotiations for new frame sizes are provided through `RequestFrameShape`. The container is responsible for drawing the frame borders that are visible when the part is active or selected, so a

container component should respond to the `AdjustBorderShape` method. A container is notified when the used shape of an embedded frame changes with `UsedShapeChanged` and may respond with code that reflows text around the embedded frame or may recalculate new frame shapes for surrounding components and then notify them with `FrameShapeChanged`.

■ **Embedding**—These methods synchronize the data content of the container and the information to be displayed by an embedded frame. Containers must also manage a list of embedded frames and return an iterator when requested.

■ **Drag and drop**—This sequence is initiated when a source part is notified, through the normal event notification scheme, of the start of a drag. The source is responsible for recognizing the selected data and obtaining an object of class `ODDragAndDrop`, adding storage content, and starting the drag. OpenDoc will assist from there, invoking the `DragEnter`, `DragWithin`, and `DragLeave` methods of the potential destination parts. You should use these methods to provide a visual indication about whether a drop is accepted. If the user drops the object, the destination part will have its `Drop` method invoked, from which the storage unit can be added to the data content, or you can invoke the draft's `AcquirePart` method to embed the data as a new component.

■ **Linking**—The potential to “link” is advertised by writing a link specification to the clipboard or to drag-and-drop objects. Destination parts recognize this and display a “Paste As” dialog. If the user desires a permanent link, the destination should invoke `GetLink` of the draft object. The source will have its `CreateLink` method invoked, where it will invoke `CreateLinkSource` of the draft and write any required data into it. On a continuing basis, the link source can update the content and flag it as changed, causing the destination's `LinkUpdated` method to be called.

■ **Undo**—OpenDoc provides for undo/redo capability through the `ODUndo` object. You are responsible for recording all undoable actions appropriate for the component in the action history. This history contains both part data and

action data. When the `Undo` method of `ODUndo` is called, the component represented by the part data will have the action data passed to its `Undo` method.

■ **Constructors/destructors**—`InitPart` and `InitPartFromStorage` are used as constructors for the OpenDoc component. `InitPart` is invoked only when the component is constructed from an empty storage unit. Reopening a document or embedding a component template will cause `InitPartFromStorage` to be passed an existing storage unit containing the component's content. Upon closing the document, the `Externalize` method will be invoked, allowing you to update the storage unit content.

An OpenDoc “Hello World” Application

It will rarely be necessary to provide specific behaviors for all of these methods. Two techniques allow generic behavior to be implemented. The first technique uses SOM's capability to subclass off of a base part. The second possibility, and likely to become the most prevalent, is to use a framework that has implemented default behavior. Two such frameworks are in development now: Apple's OpenDoc Framework and IBM's Taligent-based StarterSet. Under the covers, frameworks will provide implementations for all of these required methods and will be implemented in a way that will make it easier to customize and extend.

To give an example of the coding required to create a component, I present a “Hello World” example coded using native OpenDoc application programming interfaces (APIs) and native System Object Model interface definition language (SOM IDL). To keep the example to a reasonable length, only those methods that implement function specific to this example are included. You can find the complete sample part on the World-Wide Web at <http://pscc.dfw.ibm.com/psmag/>.

The first step in coding an application that uses SOM IDL for the class definition is to code the IDL and generate a skeleton source file using the SOM compiler. Figure 5 shows the main body of the class definition. We will add some new user methods and also override the default base class methods that we need to implement this component. OpenDoc requires

an implementation for all methods of ODPart and also the inclusion of a meta-class to be used for component registration. Since this component derives from a base part, a default implementation for all methods of ODPart has already been provided. For this example, the OpenDoc runtime will interface to our part only through the methods of ODPart, though more complex programs can publish extensions.

After the SOM compile, we have the necessary header files and the skeleton .CPP source code file. Figure 6 shows the constructor methods of our new component.

When a component is loaded into storage, one of the two constructors will be invoked. When a physical instance of the component is first created, the OpenDoc runtime invokes the component's InitPart method, annotated as (1) in Figure 6. This usually occurs when the part template is created on the desktop. Creating a template for a part consists of executing the component under the control of a registration routine, allowing the component to initialize its data content in an empty storage unit. The template created in the desktop templates folder will be the externalized storage unit.

When the template is embedded in a document, or when an existing document is later reopened, InitPartFromStorage (2) will be invoked instead of InitPart, and the component will reinitialize based upon the storage unit content.

Regardless of whether this is a new or existing component, there are certain functions programmers have found useful to implement. Section (3) shows the component obtaining the address of the ODSession object. ODSession is the main anchor for access to the OpenDoc runtime environment. We will store it persistently, because it is commonly used both for requesting the creation of service objects and for locating them at runtime. Figure 7 shows the objects that can be accessed using ODSession methods.

The second line in section (3) of Figure 6 shows ODSession being used to locate ODArbitrator. ODArbitrator manages the transfer of focus between cooperating components. Some of the specific types of focus that can be owned are:

```
interface HelloPart : BasePart
{
    void CommonInit(in ODPart partWrapper);
    attribute ODByteArray * data;

#ifdef __SOMIDL__
    implementation
    {
        functionprefix = Hello;
        metaclass = M_HelloPart;

        releaseorder:
            CommonInit,
            _get_data,
            _set_data;

        override:
            somInit,
            somUninit,
            InitPart,
            InitPartFromStorage,
            Externalize,
            DisplayFrameAdded,
            DisplayFrameRemoved,
            ReadPartInfo,
            WritePartInfo,
            Draw,
            HandleEvent,
            AdjustMenus;
    }
#endif

#ifdef __PRIVATE__
    ODSession          fSession;
    ODArbitrator       fArb;
    ODFocusSet         fFocusSet;
    ODMenuBar          fMenuBar;
    ODPopup            fPopup;
#endif // __PRIVATE__
};

#ifdef __SOMIDL__
};
#endif // __SOMIDL__

interface M_HelloPart : M_ODPart
{
#ifdef __SOMIDL__
    implementation
    {
        functionprefix = M_Hello;
        override:
            clsGetODPartHandlerName,
            clsGetODPartHandlerDisplayName,
            clsGetODPartKinds,
            clsGetOLE2ClassId,
            clsGetWindowsIconFileName;
    }
#endif
};
#endif
```

Figure 5. Using SOM IDL for the Class Definition

- Key Focus
- Menu Focus
- Selection Focus
- Modal Focus
- Scrolling Focus
- Clipboard Focus
- StatusLine Focus

As discussed earlier, if a component needs several focus types at once to effectively carry out its task, the required focuses should be requested as a set. Hello World requires the menu, keyboard, and selection focuses. Section (4) shows the code required to build the ODFocusSet object.

Section (5) shows an implementation in which the part creates its menu bar by

```

(1) SOM_Scope void SOMLINK HelloInitPart(HelloPart *somSelf, Environment
    *ev, ODStorageUnit* storageUnit,
    ODPart* partWrapper)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart","HelloInitPart");

    _InitPersistentObject(ev, storageUnit);
    _CommonInit(ev, partWrapper);

    ODSUForceFocus(ev, storageUnit, kODPropContents, kODInt1Text);
    ODByteArray * temp = CreateByteArray("Hello World", 11);
    storageUnit->SetValue(ev, temp);
    DisposeByteArray(temp);
}

(2) SOM_Scope void SOMLINK HelloInitPartFromStorage(HelloPart *somSelf,
    Environment *ev, ODStorageUnit* storageUnit,
    ODPart* partWrapper)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart","HelloInitPartFromStorage");

    _InitPersistentObjectFromStorage(ev, storageUnit);
    _CommonInit(ev, partWrapper);

    if (ODSUExistsThenFocus(ev, storageUnit, kODPropContents, kODInt1Text) {
        if (storageUnit->GetSize(ev) {
            ODByteArray * temp = CreateByteArray("",
                storageUnit->GetSize(ev));
            storageUnit->GetValue(ev, temp);
            __set_data(ev, temp);
        }
    } else {
        __set_data(ev, CreateByteArray("No Data", 7));
    }
}

SOM_Scope void SOMLINK HelloCommonInit(HelloPart *somSelf, Environment
    *ev, ODPart * partWrapper)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart","HelloCommonInit");

    if (_fSession == 0);
{
(3)     _fSession = _GetStorageUnit(ev)->GetSession(ev);
        _fArb = _fSession->GetArbitrator(ev);

(4)     _fFocusSet = _fArb->CreateFocusSet(ev);
        _fFocusSet->Add(ev, _fSession->Tokenize(ev, kODSelectionFocus));
        _fFocusSet->Add(ev, _fSession->Tokenize(ev, kODMenuFocus));
        _fFocusSet->Add(ev, _fSession->Tokenize(ev, kODKeyFocus));

(5)     _fMenuBar = _fSession->GetWindowState(ev)->CopyBaseMenuBar(ev);
        _fPopup = _fSession->GetWindowState(ev)->CopyBasePopup(ev);

        if (_fMenuBar) {
            ODPlatformMenu selMenu = _fMenuBar->CopySelectedMenu(ev);
            _fMenuBar->AddSelectedMenu(ev, selMenu);

            ODPlatformMenuItem mi;
            memset((PCH)&mi, 0, sizeof(MENUITEM));
            mi.afStyle = MIS_TEXT;
            mi.id = IDS_OPENWINDOW;
            _fMenuBar->AddMenuItemLast(ev, IDMS_SELECTED, SEL_OPENAS, &mi);
            _fMenuBar->SetMenuItemText(ev, IDMS_SELECTED, mi.id, "Window");
        }
        if (_fPopup) {
            _fPopup->RemoveMenuItem(ev, VIEW_SHOWAS, VIEW_SALARGEICON);
            _fPopup->RemoveMenuItem(ev, VIEW_SHOWAS, VIEW_SASMALLICON);
            _fPopup->RemoveMenuItem(ev, VIEW_SHOWAS, VIEW_SATHUMBNAIL);
        }
    }
}

```

Figure 6. Hello World Initialization

copying the base OpenDoc menu bar. This maintains consistency in the default menu items and their placement. Since the object is a copy, we can modify the menu contents without affecting other running parts. (Those familiar with the Macintosh desktop will notice the similarity with the

Mac's ever-present menu bar at the top of the screen.)

When you write the persistent data, I recommend that you use the predefined properties and types. Assume (if it is reasonable) that the end user may prefer to

work with a different component editor or viewer when working with this storage content. Another preferred editor will likely recognize only the standard definitions in the specification maintained by CILabs. In this example, the "Hello World" string has been stored as a standard international text string and can be viewed and modified by any other component recognizing this part kind.

A component must have a display frame in order to own display space on the screen. The initial display frame is allocated by the embedding container. The space received can be negotiated by requesting a different sized area. The request may or may not be granted. A component is not limited to one display frame.

The main function we perform in Figure 8 is to allocate a class that we have written to manage the frame's runtime information. This area is known as the PartInfoArea and is similar to the provision for window words in PM.

We create and initialize the PartInfoArea when the display frame is first added. Our FrameInfoRec class stored its address in the ODFrame object using the frame's SetPartInfo method. When the document is closed, the WritePartInfo method will be invoked to give us the chance to externalize our current status in the provided storage unit. When the document is later reopened, ReadPartInfo will be invoked, allowing the component to internalize the saved content.

To be visible, a component must, of course, be able to render its contents. To do this, the Draw method is invoked. OpenDoc requires that the imaging be done with platform-specific APIs, though future frameworks will provide an intermediate API that allows components to be platform-neutral. The example in Figure 9 takes advantage of the CFocus class, supplied with OpenDoc, to provide the required platform-specific setup.

OpenDoc provides an ODDispatcher class facility. ODDispatcher examines all events that are received by the OpenDoc PM message queue. This message queue is created by the DocShell. Based on information from ODArbitrator about which part has a specific focus, the OpenDoc dispatcher will direct the message to the appropriate

component's `HandleEvent` code. This method is implemented similarly to a PM window procedure. Figure 10 shows our event handler code.

Our part recognizes a click from mouse button 1 as a signal that the user wants to activate this part. After checking that we are not already active (by checking whether we already have the selection focus), we issue a request for our required focus types. If this method returns successfully, we can replace the current menu bar at the top of the DocShell with our own.

We have also programmed a response to the `WM_CONTEXTMENU` message. The only action we need to take is to display the pop-up menu that we built earlier. OpenDoc will take care of the rest, including dismissing the pop-up when appropriate. Finally, the example contains the code that we included to respond to the Properties menu selection.

Though we created our customized menu bar once in the initialization logic, we may require runtime modifications. `AdjustMenus` is invoked after we call `ODMenu's Display` method to allow adding check marks or graying out text.

Our last section of code, shown in Figure 11, shows our metaclass implementation that will support registration and binding. When the container calls the draft's `CreatePart` method, the `ODBinding` object will search for a component capable of handling the content.

OpenDoc uses information from several sources to make this decision.

First, OpenDoc uses the structure returned from `clsGetODPartKinds` to update the part registration database. The results of this registration record the part kinds we can edit and can be viewed using the part preferences application provided with OpenDoc. The notebook control displayed by that application was shown in Figure 3.

Second, the part will externalize its data in one or more formats when it is stored. When a user reopens the document, the OpenDoc runtime examines the types stored under the `KODPropContents` property and evaluates which editor is capable of rendering the content with the highest fidelity. The order of preference is:

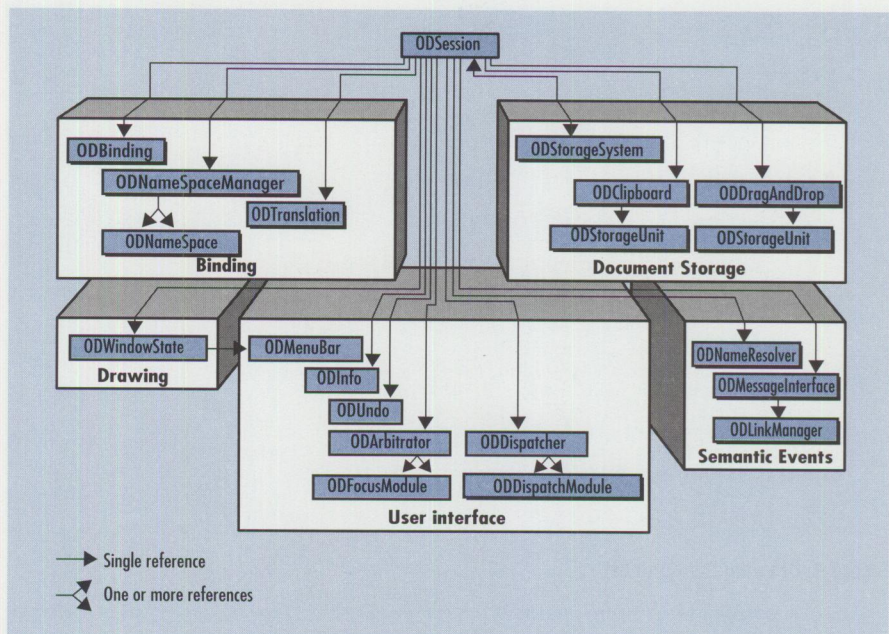


Figure 7. The ODSession Object

```
SOM_Scope void SOMLINK HelloDisplayFrameAdded(HelloPart *somSelf,
        Environment *ev, ODFrame* frame)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart", "HelloDisplayFrameAdded");

    new FrameInfoRec(ev, frame);
}
SOM_Scope void SOMLINK HelloDisplayFrameRemoved(HelloPart *somSelf,
        Environment *ev, ODFrame* frame)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart", "HelloDisplayFrameRemoved");

    if (frame != KODNULL) {
        delete ((FrameInfoRec *) frame->GetPartInfo(ev));
        _fSession->GetArbitrator(ev)->RelinquishFocusSet(ev, _fFocusSet, frame);
        ODReleaseObject(ev, frame);
    }
}
SOM_Scope ODInfoType SOMLINK HelloReadPartInfo(HelloPart *somSelf,
        Environment *ev, ODFrame* frame,
        ODStorageUnitView* storageUnitView)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart", "HelloReadPartInfo");

    return (ODInfoType) new FrameInfoRec(ev, frame, storageUnitView);
}
SOM_Scope void SOMLINK HelloWritePartInfo(HelloPart *somSelf, Environment *ev,
        ODInfoType partInfo,
        ODStorageUnitView* storageUnitView)
{
    HelloPartData *somThis = HelloPartGetData(somSelf);
    HelloPartMethodDebug("HelloPart", "HelloWritePartInfo");

    ((FrameInfoRec *) partInfo)->Externalize(storageUnitView);
}
```

Figure 8. Adding a Display Frame

- The user-specified preferred editor for the part kind
- Any editor that supports the part category
- The editor that created the part
- Any editor that can handle translated versions
- Any editor that supports the part kind
- The editor of last resort

```

SOM_Scope void SOMLINK HelloDraw>HelloPart *somSelf, Environment *ev,
ODFacet* facet, ODSShape* invalidShape)
{
>HelloPartData *somThis =>HelloPartGetData(somSelf);
>HelloPartMethodDebug("HelloPart", "HelloDraw");

>HPS hpsDraw;
>ODRect rect;

>facet->GetFrame(ev)->AcquireFrameShape(ev, kODNULL)->GetBoundingBox(ev,
>&rect);
>ODRECTL frameRect(rect);

>CFocus f(ev, facet, invalidShape, &hpsDraw);

>GpiSetColor(hpsDraw, CLR_WHITE);
>POINTL pt1 = {frameRect.xRight, frameRect.yTop};
>GpiBox(hpsDraw, DRO_FILL, &pt1, 0, 0);

>GpiSetColor(hpsDraw, CLR_BLACK);
>POINTL pt1String = {frameRect.xLeft + 20, frameRect.yBottom + 20};
>GpiCharStringAt(hpsDraw, &pt1String, __get_data(ev)->_length,
>(char *) __get_data(ev)->_buffer);
}

```

Figure 9. Drawing the Contents

```

SOM_Scope ODBoollean SOMLINK>HelloHandleEvent>HelloPart *somSelf, Environment *ev,
ODEventData* event,
ODFrame* frame,
ODFacet* facet,
ODEventInfo* eventInfo)
{
>HelloPartData *somThis =>HelloPartGetData(somSelf);
>HelloPartMethodDebug("HelloPart", "HelloHandleEvent");
>ODBooolean handled = kODFalse;
>switch (event->msg)
>{
>case WM_BUTTON1CLICK:
>{
>if ((>_fArb->AcquireFocusOwner(ev,>_fSession->Tokenize(ev,
>kODSelectionFocus))) != frame) {
>if (>_fArb->RequestFocusSet(ev,>_fFocusSet,frame) == kODTrue)
>_fMenuBar->Display(ev);
>handled = kODTrue;
>break;
>}
>case WM_CONTEXTMENU:
>{
>_fPopup->Display(ev);
>handled = kODTrue;
>break;
>}
>case WM_COMMAND:
>{
>ODCommandID command =>LONGFROMMP(event->mp1);
>switch (command)
>{
>case SEL_PROPERTIES:
>case VIEW_PROPERTIES:
>{
>ODFrameFacetIterator* facets =>frame->CreateFacetIterator(ev);
>facets->InitFrameFacetIterator(ev, frame);
>_fSession->GetInfo(ev)->ShowPartFrameInfo(ev, facets->First(ev),
>kODFalse);
>delete facets;
>handled = kODTrue;
>break;
>}
>default:
>break;
>}
>break;
>}
>default:
>handled = kODFalse;
>}
>return handled;
}
SOM_Scope void SOMLINK>HelloAdjustMenus>HelloPart *somSelf, Environment *ev,
ODFrame* frame)
{
>HelloPartData *somThis =>HelloPartGetData(somSelf);
>HelloPartMethodDebug("HelloPart", "HelloAdjustMenus");
>if (>_fMenuBar) {
>_fMenuBar->EnableMenuItem(ev, IDMS_SELECTED, SEL_OAICON, kODFalse);
>_fMenuBar->EnableMenuItem(ev, IDMS_SELECTED, SEL_OATREE, kODFalse);
>_fMenuBar->EnableMenuItem(ev, IDMS_SELECTED, SEL_OADETAILS, kODFalse);
>_fMenuBar->EnableMenuItem(ev, IDMS_SELECTED, SEL_SALARGEICON, kODFalse);
>_fMenuBar->EnableMenuItem(ev, IDMS_SELECTED, SEL_SASMALLICON, kODFalse);
>_fMenuBar->EnableMenuItem(ev, IDMS_SELECTED, SEL_SATHUMBNAI, kODFalse);
>}
}

```

Figure 10. Event Handling Code

Product Availability

OpenDoc will be packaged with a new release of OS/2 during the third quarter of 1996. Currently, Version 1 of the development and runtime environments for OS/2 are provided on the Developer Connection for OS/2, Release 9 Special Edition and on the Club OpenDoc Web site. The required compiler levels are OS/2 VisualAge 3.0 with CSD 1 or C/Set 2.01 with CSD 9. OS/2 Warp requires FixPak XR-W017.

There will be a beta release of the Windows version of OpenDoc in the second quarter of 1996, with a projected availability date of third quarter 1996. Apple has already shipped their general availability version and AIX currently has a beta version.

The Club OpenDoc Web site is an excellent source for more information about OpenDoc, including educational material, sample code, and links to other OpenDoc sites. The address is <http://www.software.ibm.com/clubopendoc>.



John Pape is an advisory development programmer in the IBM Personal Systems Competency Center, Roanoke, Texas. He supports object technologies, primarily OpenDoc.

Since joining IBM in 1978, John has worked in OS/2 application programming services and in marketing and technical support for mid-range IBM systems. He has a BS degree from Indiana University.

```

SOM_Scope ISOString SOMLINK M_HelloclsGetODPartHandlerName(M_HelloPart
                    *somSelf, Environment *ev)
{
    M_HelloPartMethodDebug("M_HelloPart","M_HelloclsGetODPartHandlerName");

    return (ISOString) "Hello";
}
SOM_Scope string SOMLINK M_HelloclsGetODPartHandlerDisplayName(M_HelloPart
                    *somSelf, Environment *ev)
{
    M_HelloPartMethodDebug("M_HelloPart","M_HelloclsGetODPartHandlerDisplayName");

    return (string) "Hello World!";
}
SOM_Scope _IDL_SEQUENCE_PartKindInfo SOMLINK M_HelloclsGetODPartKinds(M_HelloPart
                    *somSelf, Environment *ev)
{
    M_HelloPartMethodDebug("M_HelloPart","M_HelloclsGetODPartKinds");

    const char * kHelloPartKindDisplayName("Int1 Text");
    const char * kHelloPartCategory("HelloCategory");

    _IDL_SEQUENCE_PartKindInfo HelloPartInfo;

    // Create structure PartKindInfo and allocate memory for variable
    PartKindInfo * info = (PartKindInfo *) SOMMalloc(sizeof(PartKindInfo));

    info->partKindName = (ISOString) SOMMalloc(strlen(kODInt1Text)+1);
    strcpy(info->partKindName, kODInt1Text);
    info->partKindDisplayName =
        (string)SOMMalloc(strlen(kHelloPartKindDisplayName)+1);
    strcpy(info->partKindDisplayName, kHelloPartKindDisplayName);
    info->filenameFilters = (string)SOMMalloc(strlen("")+1);
    strcpy(info->filenameFilters, "");
    info->filenameTypes = (string)SOMMalloc(strlen("")+1);
    strcpy(info->filenameTypes, "");
    info->categories = (string) SOMMalloc(2*strlen(kHelloPartCategory)+2);
    strcpy(info->categories, kHelloPartCategory);
    strcat(info->categories, ",");
    strcat(info->categories, kHelloPartCategory);
    info->objectID = (string)SOMMalloc(strlen("")+1);
    strcpy(info->objectID, "");

    // copy the information into the structure
    HelloPartInfo._maximum = 1;
    HelloPartInfo._length = 1;
    HelloPartInfo._buffer = info;

    return HelloPartInfo;
}
SOM_Scope string SOMLINK M_HelloclsGetOLE2ClassId(M_HelloPart *somSelf,
                    Environment *ev)
{
    M_HelloPartMethodDebug("M_HelloPart","M_HelloclsGetOLE2Class Id");

    return (string) "";
}
SOM_Scope string SOMLINK M_HelloclsGetWindowsIconFileName(M_HelloPart
                    *somSelf, Environment *ev)
{
    M_HelloPartMethodDebug("M_HelloPart","M_HelloclsGetWindowsIconFileName");

    return (string) "";
}

```

Figure 11. Metaclass Definition Used to Register Part

Enabling Industrial-Strength OO Applications with SOM and CORBA services

This article discusses the central role that IBM's System Object Model (SOM) and Object Management Group's (OMG's) CORBA services specifications play in developing robust, reusable, distributed applications.

The first part of this article focuses on a straw man process, detailing a distributed development environment's roles, responsibilities, and contracts with each other. The article then discusses the need for a standard set of object services, such as those OMG provides. The article concludes with an architectural overview showing the primary interfaces that the various roles use, implement, install, and configure. After reading this article, you should have a good idea of what you can do today to prepare for tomorrow's distributed development environments.

Object technology has yet to fully deliver on its promises of increased productivity, large-scale reuse, and easy maintenance in developing industrial-strength software applications. The delay is not due to a limitation of the technology itself, but rather how it is used by mainstream practitioners. In short, few of us exploit the additional encapsulation afforded by objects. Luckily, the rising popularity of three-tier software architectures, with their dependency upon distributed object services, has forced us to start moving in the right direction.

Why Distributed Objects?

The notion of a distributed application goes beyond simple client/server separation to encompass the idea of distributed development, separating the

is the same—there have to be two things: a development process and a mechanism that allows separately developed (i.e., binary) objects to be integrated at runtime. IBM's System Object Model (SOM) provides the mechanism, while the Object Management Group's (OMG's) CORBA services specifications provide a standard platform of object services around which a development process can be built.

Defining the Contracts

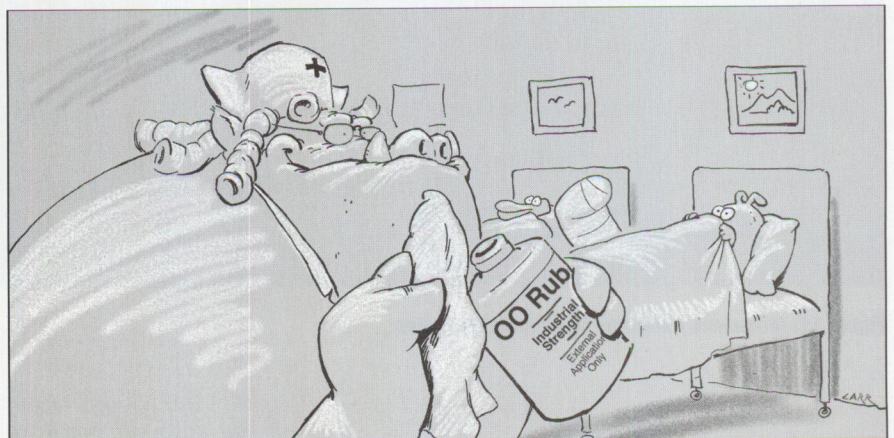
Figure 1 shows a straw man development process for distributed applications, showing at a high level the roles, their primary areas of expertise (i.e., concerns), and the work products that they are responsible for developing.

Roles and Responsibilities

In Figure 1, the *end-user* (EU) role is included for completeness. End users are primarily concerned with their own job responsibilities, and they use distributed objects to help manage job-related tasks.

Geoff Hambrick
IBM Corporation
Austin, Texas

concerns of the individuals involved in software development and their various roles, so that each can work in parallel (or incrementally/iteratively over time). The practical implication for either interpretation



The three primary roles taken on by those responsible for software development include:

- **Object Providers (OPs)** build new classes of objects, capture the business logic of a domain (e.g., insurance, banking, or travel), and use a standard set of object services (to be described in detail later in this article).
- **Service Providers (SPs)** build classes of objects that implement these standard services on a particular set of platforms and technologies (such as DB2 or Lotus Notes).
- **Application Assemblers (AAs)** create distributed application objects by installing and configuring business objects with object services to meet the operational requirements of the customer environment.

The most important idea here is that a set of standard object services serves as the basic contract between the roles. Although the straw man process proposed here is abstract enough to allow for any object services standards to be used, I will focus on the object services defined by the Object Management Group as part of its CORBAServices architecture.¹

Object Services

For this discussion, it is useful to divide object services into the following three categories: explicit services, implicit services, and enabling services.

Explicit services are used by an OP in developing business objects, implemented by an SP, and configured by an AA. This category includes services already adopted by OMG, such as naming, query, events, and properties, as well as those not yet adopted, such as collections. These explicit services form the basis of the contracts between the three roles.

Implicit services are those that the OP can assume to exist when developing business objects, such as persistence, transactions, concurrency, and security. (Security has yet to be formally adopted by OMG but was recently recommended for adoption by the OMG Technical Committee.) Although these services are

¹ Object Management Group, *CORBAServices: Common Object Services Specification*, Revised Edition, March 31, 1995, OMG Document Number 95-3-31.

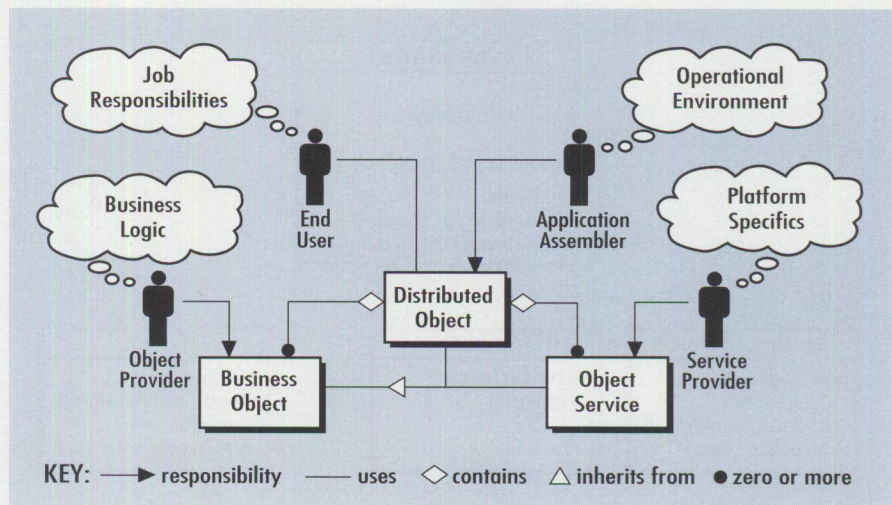


Figure 1. Distributed Development Roles, Responsibilities, and Work Products

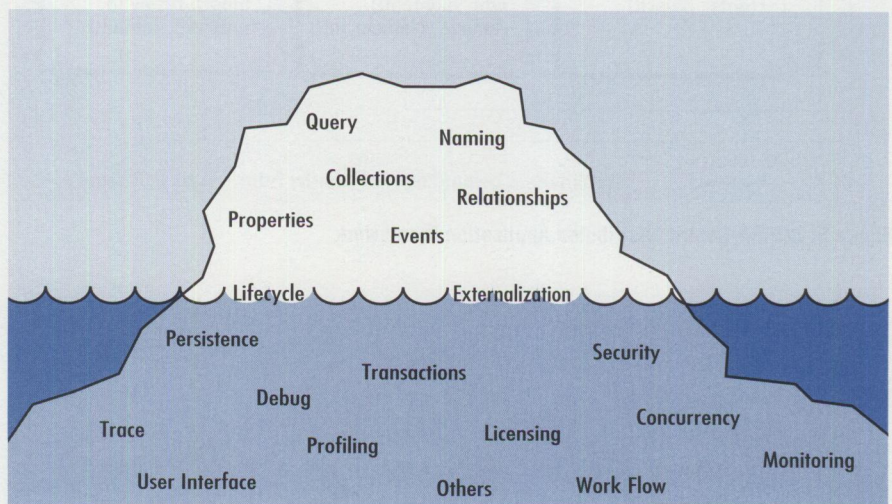


Figure 2. The Distributed Object Iceberg

only indirectly used by an OP during development, they are implemented by the SP and configured by the AA.

Enabling services are a class of explicit services implemented by an OP, configured by an AA, and used by an SP to tie implicit object services to business objects in a domain-independent manner. These services, such as lifecycle and externalization, are the only ones requiring the OP to implement specific interfaces.

The Distributed Object Iceberg

An interesting way to visualize the relationship among these three categories of services is to imagine a distributed object as an iceberg (see Figure 2). A distributed object combines one or more business objects (implemented through services in the tip of the iceberg) with one or more

object services (implemented through the massive, invisible part underneath the surface). Enabling services (which serve as the waterline and are visible to and implemented in part by all) are used when required to tie both domains together.

Distributed Application Framework Architecture

The previous discussion provided a conceptual background to understand the CORBAServices at a high level; however, you will need specific details (such as a class hierarchy and relevant interfaces) to really begin developing distributed applications. Figure 3 is a straw man for the rest of this article. The objects in Figure 3 show the more important CORBA-compliant interfaces they use and/or expose (for others, see the CORBAServices

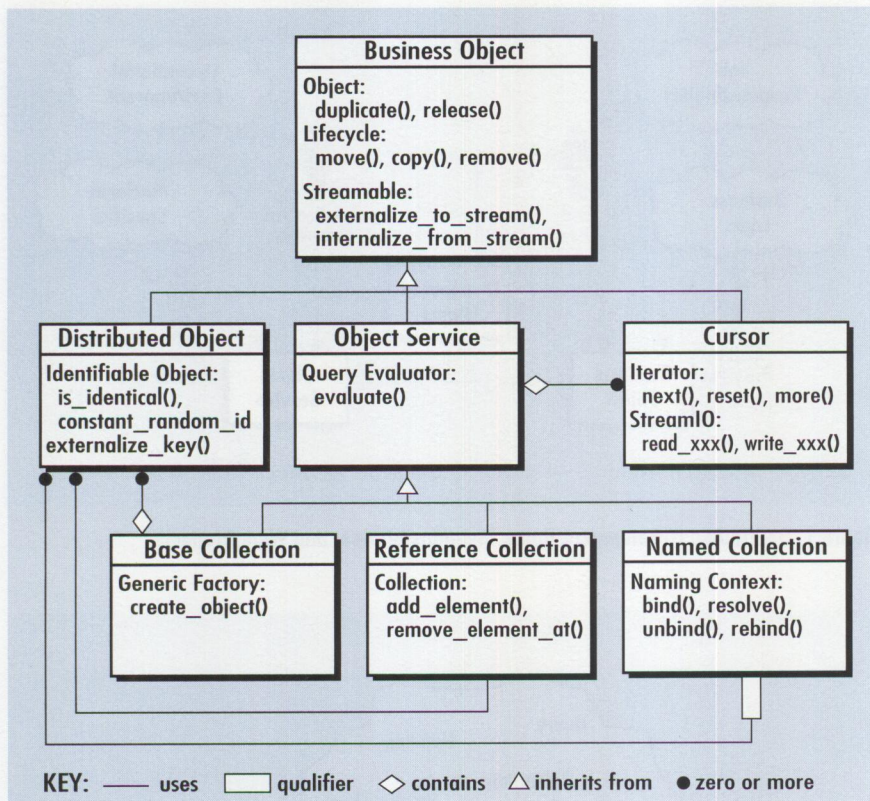


Figure 3. CORBA-Centric Distributed Application Framework

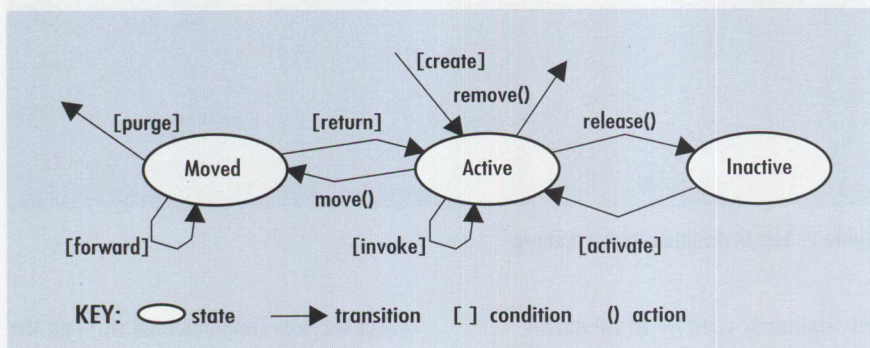


Figure 4. Business Object Lifecycle

specifications), as well as some fundamental relationships among these objects. I'll briefly describe each.

Business Object

In general, *Business Objects* (BOs) will be created by the OP; however, as the framework shows, a Distributed Object (DO) is a subclass serving as the first "real" manifestation of a BO in a distributed application, using the various types of Object Services. The primary reason for this arrangement is to allow the AA to "insert" implicit services (such as identity, reference counting, persistence, and/or transactions) into a BO, transparent to the OP. In practice, whenever an OP thinks that he or she is

working with a BO, he or she is actually dealing with a DO. (I'll discuss DOs and their relationship to BOs and Object Services in more detail in the next section.) The rest of this section describes the services that are associated with each BO.

- `duplicate()` is used by "good-citizen" BOs (i.e., those that are coded by the OP according to good programming practices) when handing out references to other BOs, so that the Base Collection can reference-count the associated DO if desired.
- `release()` is used by good-citizen BOs when they are done with a reference to

other BOs, so that the Base Collection can garbage-collect the DO if desired. This should not be confused with the `remove()` method.

- `move()` is used to transfer a BO's "ownership" (and identity) and associated DO from one Base Collection to another.
- `copy()` is used to create a new BO and DO initialized from the BO called. *Note:* The new object's identity depends upon the Base Collection in which it is created.
- `remove()` is used at the end of an object's logical lifecycle to completely destroy the BO and associated DO.
- `externalize_to_stream()` is used to export the essential internal state of a BO so that it can be made persistent, passed by value, cached, moved, copied, or, in general, used by any services needing access to the BO's state.
- `internalize_from_stream()` is used to import the essential internal state of a BO as a consequence of restoring its persistent state or otherwise initializing it from a move, copy, cache, etc.

You will notice that these are enabling services—that is, OPs will implement and use them to help move the various BOs of their domain through a stylized lifecycle (see Figure 4).

A given Distributed Object extends most of these methods to add implementation-specific behaviors that are used with various Object Services.

Distributed Object

An AA configures *Distributed Objects* by mixing together a Business Object provided by an OP and a Distributed Object implementation provided by an SP. For example, an OP may provide an Employee Business Object, while an SP may provide a DB2 Distributed Object. The AA may configure them into a DB2 Employee Distributed Object.

As mentioned before, a DO is used to transparently add services to a BO, especially identity, which the SP provides with a Base Collection (a kind of Object Service described in the "Base Collection" section further in this article). The additional services associated with a DO to be described here are usually implicit and not often used directly by an OP, although there are exceptions. Instead, an SP uses them in implementing Object Services,

such as Reference and Named Collections (described later in the "Reference Collection" and "Named Collection" sections).

A good way to visualize this relationship is by imagining a Base Collection as a circuit board, with the DO as an intelligent socket (in that it holds identity and other states) and the BO as the chip that plugs into the socket (see Figure 5).

Messages first must pass through the system backplane into the circuit board and through the socket before they are propagated to the chip itself; therefore, any number of additional implementation-dependent relationships and behaviors, such as the following, can be inserted without involving the chip designer (in this case, the OP):

- `is_identical()` is used to determine if two references refer to the same logical DO (even though they may be physically replicated).
- `constant_random_id` is a read-only attribute used to quickly determine if references are different, because this ID can be cached with local replicas (or proxies as they are commonly known); however, having two identical values is no guarantee that the references are to the same logical DO (use `is_identical()` to be sure that two references are equal when two `constant_random_ids` are equal).
- `externalize_key()` is used to write an external form of its key into a stream, usually during a `write_object` call on a `StreamIO` or `Cursor`.²

Object Service

The SP creates *Object Services*, whose main purpose in this level of the hierarchy is to serve as a common base class for the concrete types (such as Base Collection, Reference Collection, and Named Collection) in order to take advantage of any implementation details, yet allow for federation. In general, an Object Service's purpose is to provide access to one or more associated Distributed Objects via the following interface: `evaluate()` takes a query string (of various syntaxes³) and returns a CORBA "any" type.

However, for purposes of this straw man architecture, I recommend that a `Cursor` be returned.

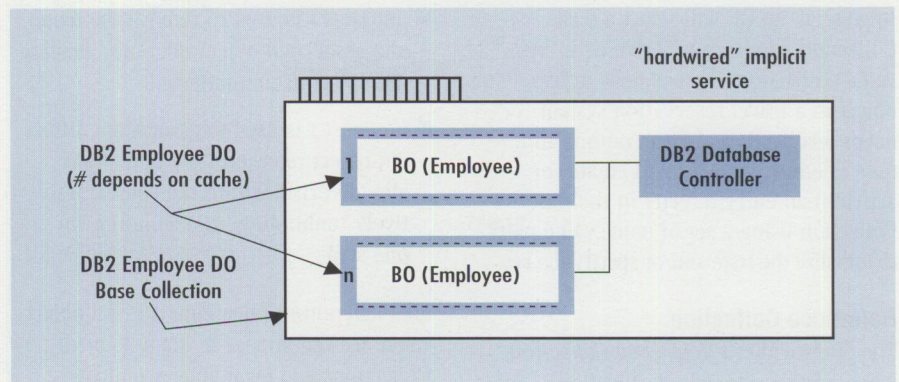


Figure 5. Base Collection Circuit Board

Cursor

Object Services and *Cursors* are meant to be developed in pairs (to take advantage of any underlying implementation for query evaluation), just as a Distributed Object is meant to be developed in conjunction with a Base Collection (so that the key can be implemented efficiently).

Regardless of what you call it, you need an object to be returned that is flexible enough to handle the wide variety of information that can be "selected" in an SQL-like query string:

- Object references only (e.g., `select &` to make an associated Reference Collection)
- A subset of data from the matching entries (e.g., `select name phone` for a report or view)
- A copy of the matching entries (e.g., `select *` for an archival snapshot)

The following methods on `Cursor` are designed to support this flexibility:

- `next()` is used to position the cursor at the next entry of selected data
- `reset()` is used to position the cursor back to the first entry
- `more()` is used to check the cursor for more entries that can be processed
- `read_xxx()` is used to read the

next data item from the entry, where `xxx` is an IDL data type or an object reference (the latter is read via `read_object()`)

- `write_xxx()` is used, when a cursor supports update or insert, to write the next data item to the entry

The `StreamIO` functions (`read_xxx()` and `write_xxx()`) eliminate the need for "any" processing, since the type is explicit in the function name.⁴ They also allow use of `internalize_from_stream()` or `externalize_to_stream()` with the cursor as the source of the data in the copy scenario described.

In this manner, a `Cursor` can be viewed as a "window" over the result data set of a query, with the `Iterator` functions providing sequential access to entire entries and the `StreamIO` functions providing sequential access to the individual data items within the entries.

Base Collection

An SP provides a *Base Collection* primarily for identity; however, it is also the means by which other implicit services (such as persistence, transactions, concurrency, and security) are added to Distributed Objects. For example, an SP who is an expert in DB2 databases may develop a DB2 Base Collection that not only provides identity (through the database table name and key fields) but also transactions, persistence, and security.

² The `externalize_key()` is not part of the OMG services but was deemed necessary to enable identity within the business object lifecycle described in Figure 4 without explicitly architecting a key format. In particular, activating an object entails using a previously externalized key during the `create_object()` operation of the containing Base Collection.

³ Object Management Group, *Object Query Service Joint Submission*, OMG Document Number 95-1-1.

⁴ An "any" type returns a type code and an associated value. In practice, the type code is interpreted via a "case" statement unless the type is explicitly known (as in this example). The use of streams is more efficient when the type is known because no "any" structure is created and passed.

In general, an OP will select a Base Collection wherever a BO needs a “by-value” collection. For example, a Trip object in a travel reservation system needs a collection of Reservations and uses `create_object()` to create (or activate) an entry directly in the Base Collection using a set of name-value pairs to initialize the state and/or specify the key.

Reference Collection

The SP provides a *Reference Collection* when a BO needs only a collection of references to other BOs. For example, a Travel Service might only need a list of references to Reservations. The following methods are needed:

- `add_element()` is used to add a reference to an existing object to the collection of references.
- `remove_element_at()` is used with a Cursor to take elements out of the collection.

The ramification is that the objects maintained in a Reference Collection must already exist in a Base Collection (already described), although they might have been located through a Named Collection (described next). It also means that a given object can appear in any number of Reference Collections, or even multiple times within the same Reference Collection (which is why the `remove_element_at()` method needs to use a Cursor to point to the specific entry). In any event, the object references become part of the essential state of the Reference Collection itself (see the streamable functions in the previous BO description).

Named Collection

The SP provides a *Named Collection* when a BO needs to have a uniquely qualified (named) list of references to other BOs. For example, a Trip might only support one Reservation of each type, such as Car Rentals, Airlines, Hotels, etc. Named Collections support the following methods:

- `bind()` is used to associate a unique (within this collection), human-understandable name to an existing object and, in effect, add to the collection of named references.
- `resolve()` is used to look up a name and return the associated object reference (if the name is found within this context).

- `unbind()` is used to remove the name and associated object reference, leaving the object itself unaffected.
- `rebind()` is used to associate a different object reference to a name that already exists within the context, effectively “unbinding” and “binding” in one step.

Like Reference Collections, a given object reference can appear in many Named Collections and even appear multiple times in the same one (if different names are used to bind it). Further, both the name and the object reference to the bound objects are part of the Named Collection’s essential state.

The Development Process

Now that we have defined an implementable architecture, the next step is to understand the dynamics of how to use it to develop a distributed application. Space limits me from taking an example through the entire process; however, I will overview the major paradigm shifts.

The Role of SOM and Binary Reuse

SOM is an excellent mechanism to develop objects in the language and compiler of your choice (currently C and C++) and integrate them at runtime. If your shop does all development with a single compatible set of compilers, then this feature may not seem useful to you; however, over time and with future releases, even compilers within the same family can become incompatible. Using SOM now can help keep you from having to “recompile the world” later. Also, if your shop cannot afford (or lacks the expertise) to develop some of the more complicated object services (such as transactions and concurrency), then you will appreciate the ability to use the objects you develop with binary objects developed outside of your control.

To use SOM with this architecture, simply make `SOMObject` the “top” of your inheritance hierarchy by adding to the list of Business Object “parent classes.” This subtle movement away from language-centric programming represents the first paradigm shift.

The Need for IDL

The second paradigm shift you must make when developing distributed applications is to start with the interfaces to Business

Objects, that is, start with CORBA interface definition language (IDL). Of course, you can use a tool to help you graphically draw your Business Objects and their relationships, methods, and attributes, then automatically generate the IDL.

IDL is necessary for your local compiler to link, load, and execute the binary objects supporting the interfaces described. Usually IDL is run through an “emitter” (i.e., a code generator) to generate headers in your development language, so that your Distributed Objects can “naturally” invoke methods on objects developed by others (down the hall, across the world, or even by you from an earlier point in time). Thus, even though a Business Object or an Object Service may be developed in C, your C++ Distributed Objects can treat them as if they were implemented in C++.

Coding Enabling Services in All BOs

The third major paradigm shift you’ll make concerns how you develop your Business Objects (even if they are Distributed Objects, Object Services, or Cursors). You should override the enabling services associated with BOs in the previous section:

- `duplicate()` is usually not overridden in a BO, while DOs that reference-count will often increment the count prior to calling the parent BO.
- `release()` is implemented in the BO by propagating a `release()` to all other BOs to which it maintains references. A DO that reference-counts will often decrement the count and only propagate the call to the parent BO when the count reaches zero.
- `move()` is often implemented by default in the BO to externalize itself to a stream, use the stream to initialize a newly created DO in the Base Collection, then remove itself. Some DOs will keep a forwarding reference to the new entry for a period of time.
- `copy()` is similar to move, except that the DO need not worry about forwarding, because the state of the object does not change.
- `remove()` is implemented in the BO by propagating a `remove()` to all “sub-objects” (i.e., those owned by this BO), and a `release()` to all others. A DO will often set a state indicating that the

object has been deleted, so that the key does not get reused.

- `externalize_to_stream()` is implemented in a BO by writing to the stream passed in all the non-derivable internal states in a form that is not likely to change. Primitive attributes are written using their corresponding IDL types (float, long, string); object references are written using `write_object()`; sub-objects are written recursively using `externalize_to_stream()`; and sequences (although for maximum configurability by an AA, writing a reference to one of the collections described above should be used instead) are written sequentially with a `write_end()` after the last entry. DOs only have to externalize a state, according to these rules, that isn't already maintained in the Base Collection's underlying datastore (the `externalize_key()` method follows these rules as well).
- `internalize_from_stream()` is implemented the same way as the externalize, except that the reads must exactly match the writes (and a `read_end()` should be checked prior to reading data entries in the sequence).

Providing an Essential Data Schema

The fourth paradigm shift is that you should provide an attribute-only "map" (or schema) of the data written during externalization. This map, which is called the essential data schema (EDS), enables the AA to transform the stream data into other forms such as existing database schemas. If this data is not provided, then the AA is limited in the following ways:

- Persistence mechanisms must support "opaque" fields (uninterpreted strings of binary data).
- All queries are required to activate each object during evaluation.
- No indexing on given fields can be supported.
- User-interface panels must get their data from active objects.

Unfortunately, IDL does not support a "schema" section, so a naming convention is required. I recommend that the IDL for a given BO have `_BO` appended, with the associated essential data schema having `_EDS` appended. For example, the two IDL interface definitions provided for an

```
empF = (BaseCollection *)DNC -> resolve("Employee");
emp  = (Employee_DO *)empF -> create_object();
emp  -> name    = "John Jones";
```

Figure 6. Factory-Centric Sample⁵

Employee class should be `Employee_BO` and `Employee_EDS`.

For a DO, a convention such as `_DO` and `_KDS` (for key data schema) should be followed so that the AA can use the schema of the key to map to underlying databases and user interfaces.

OPs and SPs who want to maintain some proprietary data while exposing other, more open fields can lump all the proprietary data together into one attribute (such as `other_data`) or simply give the individual fields meaningless attribute names that are not explained in the user documentation. The former allows for more opacity (since the types are not known) and the latter for more efficiency (because there is no need to first buffer the fields, which can still be indexed, queried, and so on).

Coding Domain-Specific Methods Using a Factory-Centric Programming Model

The basic paradigm shift here is that, by using the explicit interfaces of object services and enabling interfaces of other business objects, your object will not only be a good citizen in a distributed environment, but it will also be far more configurable than if it were to directly implement dependencies upon specific implicit services.

Beyond using these services, you should no longer use the create operation of your favorite development language to create new business objects. Instead, you must adopt a *factory-centric* programming model using the following steps:

1. Find a "factory" (Base Collection) that can create/activate the type of BO you require.

2. Send this factory the `create_object()` method as described in the Base Collection section.
3. Use and/or store this reference as part of your BO's essential data.

Specifically, each server process will have access to a Named Collection called the Distinguished Name Context (DNC). It will have various Base Collections bound to it by the AA. Therefore, the steps above can be as simple as writing the few lines of code shown in Figure 6.

By using this factory-centric programming model, the AA can insert the implicit services that you as a BO programmer need not worry about. This does not mean that you cannot use a native create operator to manage sub-objects that you do not want the AA to know about; however, you would then be responsible for maintaining their integrity (usually by simply externalizing their essential state as part of your own).

Once your BO references another BO (with associated DO), its programming model described in the architecture section applies as-is. It is not repeated here.

Indicating Dependencies

The final paradigm shift is the way that BO dependencies are communicated to the AA in the absence of a formal IDL section to show dependencies between classes. As in the previous subsection, the OP determines the name and the expected class managed by the Base Collection that is to be bound into the DNC by the AA.

Again, I recommend using a naming convention and a special dependencies file (`<module>.dep`). This file contains a list

⁵ Technically speaking, the `resolve()` interface does not directly take a string as the "name" input parameter; it takes a sequence of structures of name/extent strings instead. (See the CORBAServices specifications for exact details.) Also, the DNC could be replaced with a "FactoryFinder" (from the CORBAServices Lifecycle specification). Choosing a NameContext was more consistent with the notion that the OP would define the name used, the SP would implement a service for finding it, and the AA would bind the appropriate "factory" (i.e., BaseCollection) into it.

FREE

Product Information Index

STOP!

Have you used the reader service card to request fast, free information about the products and services advertised in *Personal Systems*?

NO? With the heavy traffic of new technology to choose from in the personal computer market, you need to know about all the most recent developments.

Caution.

1. Use the advertiser's index to get the reader service numbers of the products and services for which you want to receive literature.
2. Circle the same numbers on the reader service card and fill out the necessary information.
3. Drop it in the mail (at no charge!), and we'll give your request the green light!

YES.
Smart move.

Reader Service Number	Company	Page #
8	BT&T Consumer Technology	14
3	Brooktrout Technology	10
18	ChipChat Technology Group	3
5	Cirrus Technology	8 & 12
1	Conner Tape Products Group	6
13	Coriolis Group	17
23	DFI	11
22	Hilbert Computing	8
12	IDG Books	16
21	Indelible Blue	10
14	John Wiley & Sons	17
4	Lexmark	10
24	Micro Channel Developers Assn.	13
20	Microrim	9
11	Percussion Software	16
17	Pinnacle Technology	1
2	Polygon Wire Management Systems	8
7	Rhintek Computer Engineering	12 & 14
19	Softmart	7
15	SofTouch Systems	Cover 2
16	Stardock Systems	Cover 4
10	Sybase, Inc.	15
6	SYNTEGRATION	14
9	Volant Corporation	15

of names and classes and, if the class is one of the Object Services types such as Base Collection, Reference Collection, or Named Collection, it contains the class of entry maintained, referred to, or bound (see Figure 7 for an example).

You should note that many BOs will maintain references to entire Base Collections, Reference Collections, or Name Collections. Not only does this reduce the amount of code that the OP has to write (to query, insert, delete, and externalize/internalize), but it also permits the AA a high degree of configurability, since an AA can bind the same collections into the DNC under many names (as long as the returned class is the same).

For example, the AA could bind the factory for both planned and completed trips to the same Base Collection or two different ones with radically different qualities of service (one stored in flat files, another in a DB2 database) without change to the Member or Base Collection class.

More About Services

Hopefully, this overview helps to illustrate the kind of flexibility a distributed application development environment offers. Following are some important points to remember about the services discussed:

- The word "implicit" in object services refers only to their use by the OP when developing business objects—not whether the service is visible to an SP or AA. For this reason, services such as user interface and persistence are considered to be implicit.
- Object services are considered a subclass of Business Object, to allow the AA to flexibly manage them without architecting how their keys should be implemented. In general, all Business Objects should be written to be independent of identity so that they can be used in many different contexts. This philosophy extends to Object Services and Cursors (although the latter are often implemented without identity).
- There are many other implementation choices, even within a single service such as security (e.g., two- versus three-party authentication, public versus

```
TRS::Member::planned_trips BaseCollection TRS::Trip
TRS::Member::completed_trips BaseCollection TRS::Trip
TRS::Member::profiles ReferenceCollection TRS::Profile
TRS::Trip::reservations NamedCollection TRS::Reservation
TRS::Reservation::trip TRS::Trip
```

Figure 7. Example of Contents of the TRS.dep File Indicating Dependencies

private key encryption, capability versus ACL-based authorization). Therefore, object services should be thought of as relatively open-ended categories.

- The list of services, especially the implicit ones, will continue to grow as technology evolves. Therefore, you need to consider such non-standard services as Trace and Debug, which are sure to become widely available as part of a distributed application development framework.
- There is no absolute requirement that the SP implement these services using object technology nor that these services must expose "standard" interfaces beyond those described. This point is made not only to reinforce the idea that objects are encapsulated but also to urge you not to wait for an outside company to provide these object services for you.

What Can You Do Now?

Without waiting for another company to provide you with a set of object services, what can you do today to enable robust, reusable distributed applications?

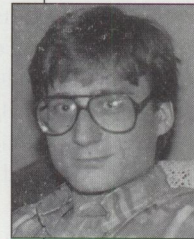
1. Follow the cookbook described here to separate the concerns of OPs, SPs, and AAs for code you develop.
2. Expect those who deliver code to you to recognize these separate concerns.
3. Look for code in your shop that provides similar function to the object services described above and "wrapper" (i.e., build object layers around) them using SOM to support the interfaces defined.

In the end, you will find that having a well-defined, distributed architecture will save you development time and effort, even if you must code everything yourself, because the work of defining the interfaces has already been done for you by OMG, and the mechanism for making binary objects "distributable" is already provided by IBM's SOM technology.

For more details about the exact signatures and other methods associated with the CORBAServices specification, so that you can extend this straw man architecture or develop your own, see Object Management Group's *CORBAServices: Common Object Services Specification*.¹

Acknowledgments

I want to especially thank the following people for their efforts during the past year to help me understand the role of object services in a robust, reusable, distributed application: George Copeland, Randy Fox, Jim Sides, Rob High, Don Ferguson, Jim Rayfield, Eric Herness, Charlie Redlin, Frank Malin, Rimas Rekasius, and Tony Wells.



Geoff Hambrick is an advisory programmer within the IBM Software Group Technology Center in Austin, Texas. He is currently the technical lead on advanced software development projects that identify, select, and/or create key software technologies and architectural frameworks that support the Open Blueprint strategy. Previously, Geoff was technical lead in developing reference implementations of CORBA-compliant Object Services and making their transition into product development groups. He earned an outstanding technical achievement award for his work as a consultant helping first-time users of object technology.

Geoff has presented object technology at several major conferences, and, in an upcoming book, he has co-authored a chapter about practical applications of end-to-end OO development methodologies. Before joining IBM in 1989, Geoff worked for Texas Instruments. He earned a BA degree in Computing Science from the University of Texas in 1981. His Internet userid is geoff@austin.ibm.com.

Geoff has presented object technology at several major conferences, and, in an upcoming book, he has co-authored a chapter about practical applications of end-to-end OO development methodologies. Before joining IBM in 1989, Geoff worked for Texas Instruments. He earned a BA degree in Computing Science from the University of Texas in 1981. His Internet userid is geoff@austin.ibm.com.

SOM Language Neutrality: A VisualAge for Smalltalk Perspective

VisualAge for Smalltalk (VAST) is a premier workstation application development tool. This article addresses the use of VAST to provide a graphical user interface front end to the valet parking system (VPS) application described in Rick Weaver's "The IBM System Object Model—The Wave of the Future (and Now!)" article. The simplicity of the overall implementation, the speed with which such an extension can be developed, and the strength of the VAST technology to support this effort should be apparent to most application development professionals.

Smalltalk provides one of the most productive application development environments available. Its productivity is extended even further by adding VisualAge's visual development enhancements. VisualAge for Smalltalk supports IBM's OS/2 and AIX operating systems, as well as Microsoft Windows as primary VisualAge application execution environments. It also provides a variety of additional product support, including special connection features for AS/400 and IMS.

VisualAge's SOMsupport feature provides an interface for using objects implemented with SOM in a VisualAge application. With this interface, VisualAge applications can create instances of SOM objects and send them messages. SOM objects can be developed in any language that has SOM implementation bindings.

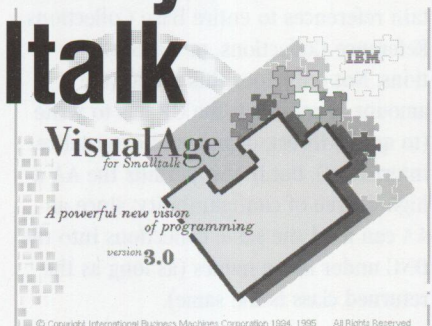
The Valet Parking System

The valet parking system (VPS) application is implemented in C++. This sample application is designed to support a customer bringing in a car to be parked by a valet in a garage or having a parked car retrieved by the valet. The classes implemented within VPS are Valet, Ticket, Car, Garage, and TktBookS.

Tom New
IBM Corporation
Roanoke, Texas

The VPS user interface was developed using VAST. Shown in Figure 1, the VAST front end uses the attributes (data) and methods (functions) associated with the C++ VPS, which are shown in Figure 2.

The other objects (Garage and TktBookS) are hidden from the VAST front end application by the Valet object. These VPS C++ objects are accessed through the SOMsupport feature of VisualAge for Smalltalk, Version 3.0.



VAST SOMsupport

IBM Smalltalk and VisualAge for Smalltalk provide usage bindings for SOM and Distributed SOM (DSOM). VisualAge for Smalltalk, Version 3.0 for OS/2, is used for the sample application interfacing to the C++ valet parking system SOM application. The VisualAge for Smalltalk SOMsupport feature used to interface to VPS is available in both OS/2 and AIX.

SOMsupport provides facilities for a developer to easily build and use SOM Smalltalk wrapper classes visually in the VAST Composition Editor. When you add a wrapper object to the Composition Editor, it appears as a non-visual part. Attributes and actions on the VisualAge part's public interface represent the SOM object's attributes and methods.

SOMsupport for VisualAge requires either the SOMobjects Base Toolkit, Version 2.1, or the SOMobjects Developer Toolkit for OS/2, AIX, and Windows, Version 2.1, on your workstation. VisualAge provides the SOMobjects Base Toolkit along with the

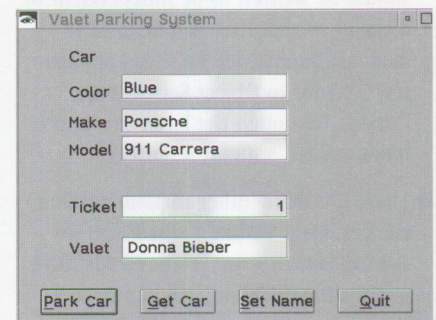


Figure 1. Valet Parking System

VisualAge SOMsupport feature. This Base Toolkit is a subset of the SOMobjects Developer Toolkit. The SOMobjects Developer Toolkit should be used when collection, persistence, or replication framework support is required. This Toolkit is also needed to provide all required files to develop your own SOM classes in C, C++, or COBOL.

SOMsupport enables you to use the SOM and DSOM classes described in the interface repository (IR) in an application by allowing you to generate Smalltalk code to access them. If your SOM or DSOM classes point to a DLL, the generated code (called a wrapper) enables your application to use programming logic in the DLL, letting the application create and send messages to local or remote SOM objects.

SOMsupport consists of Smalltalk classes, some of which are actually wrappers for SOM objects, and comprises the following areas:

- SOM runtime library
- SOM frameworks (DSOM, interface repository, metaclass)
- SOM Smalltalk constructor
- SOM samples
- SOMsupport migration facility (where VAST Version 2.0 SOMsupport was used)

VAST VPS Environment

Now let's begin building the GUI front end to the valet parking system. I used the following hardware and software for this process:

- OS/2 Warp Version 3.0
- VisualAge for Smalltalk, Version 3.0 for OS/2
- VisualAge for Smalltalk, Version 3.0 SOMsupport feature
- SOMobjects Developer Toolkit, Version 2.10, CSD level SM21002
- SOMobjects Collection Classes, Version 2.01.5, CSD level SM20012
- SOMobjects Interface Repository, Version 2.01.5, CSD level SM20012
- PS/2 Model 90 XP486 with 32 MB of RAM

Class	Attributes	Methods
Valet	valetName	parkCar, retrieveCar, the getter and setter for valetName
Car	color, make, model	getters and setters for the attributes
Ticket	ticketNum	getter and setter for ticketNum

Figure 2. VPS Attributes and Methods

Loading SOMsupport

First, install the SOMsupport feature and have a SOMobjects toolkit on your system. (The VPS sample application requires the SOMobjects Developer Toolkit because it uses SOM collection classes.) You must also set environment variables for SOM and DSOM. *Note:* A VAST application using SOM requires SOM CSD 212.

To install SOMsupport, follow the product's installation instructions. After you install SOMsupport, set the environment variables and reboot, then load the SOMsupport feature into your VAST image. From the "Smalltalk tools" menu in the System Transcript window, select "Load Features." Then select "SOMsupport, Base Feature." Although it's not specifically required for this sample application, you can select "SOMsupport Samples." After loading the features, select "Yes" at the "Save image" prompt.

Setting Up the Interface Repository

To wrapper a class, SOMsupport must be able to access its SOM interface definition language (IDL) description. These SOM IDL class descriptions must be available in a SOM IR (interface repository), which you define by editing the CONFIG.SYS file or by issuing the SET SOMIR command.

To set up the VPS IR requirements, follow these steps:

1. Load valet.ir onto a hard drive. (I loaded it into E:\valet.ir.)
2. Enter SET SOMIR=e:\valet.ir; at an OS/2 command-line prompt, or add it to the SET SOMIR= statement in the CONFIG.SYS file. If changing SET SOMIR in the CONFIG.SYS file, you'll need to reboot OS/2.
3. Ensure that the valet.idl, car.idl, and ticket.idl files can be accessed by entering irdump Valet, irdump

Car, and irdump Ticket respectively at the command prompt (Valet, Car, and Ticket are case sensitive here).

VPS DLL Setup

VisualAge for Smalltalk must be able to access the VPS DLLs, as well as other DLLs used by the VPS; therefore, you need to copy the following DLLs to the VISUALAG\DLL directory where the SOMsupport samples are also loaded:

- car.dll
- dde4mbs.dll (a SOM DLL)
- garage.dll
- ticket.dll
- tktbooks.dll
- valet.dll

Additionally, since the SOMobjects Collection Framework uses somuc.dll within the VPS application, it needs to be accessible through a directory in the CONFIG.SYS LIBPATH statement.

VAST Valet Application Development

Start VisualAge for Smalltalk by double-clicking on the VAST icon. If you know you're going to work with SOM or DSOM objects, start VAST from an OS/2 command prompt to redirect your standard output (stdout) to a file. Change to the VISUALAG directory, and enter abt somstdo.txt (or whatever filename you desire). Then, if your development image ends processing due to a SOM fatal error, you can examine the file for information about the error.

Wrapping SOM Classes

VisualAge Quick Start was the starting place for developing the Valet application—it shows up after you double-click on the VisualAge for Smalltalk icon (see Figure 3). As previously noted, to use a SOM or DSOM object in a VAST

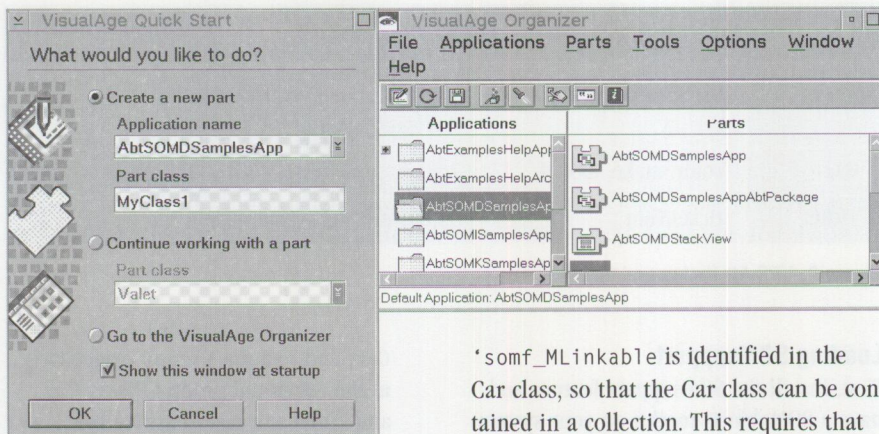


Figure 3. VisualAge Quick Start

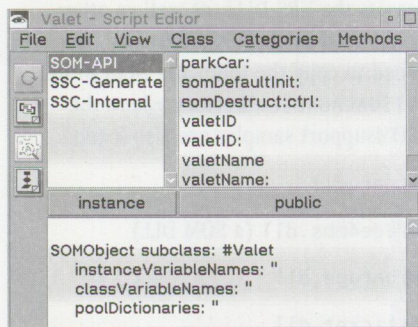


Figure 4. Generated Method Wrappers for Valet Part

application, you need to generate Smalltalk code known as a wrapper. When you installed SOMsupport, VAST added the "SOM Wrappers" sub-menu choice to the VisualAge Organizer. Select "Parts -> Generate -> SOM Wrappers..." to open the Create SOM Wrappers window.

Now you need to wrapper the required SOM classes. For the VPS application purposes, the SOM Collection class for

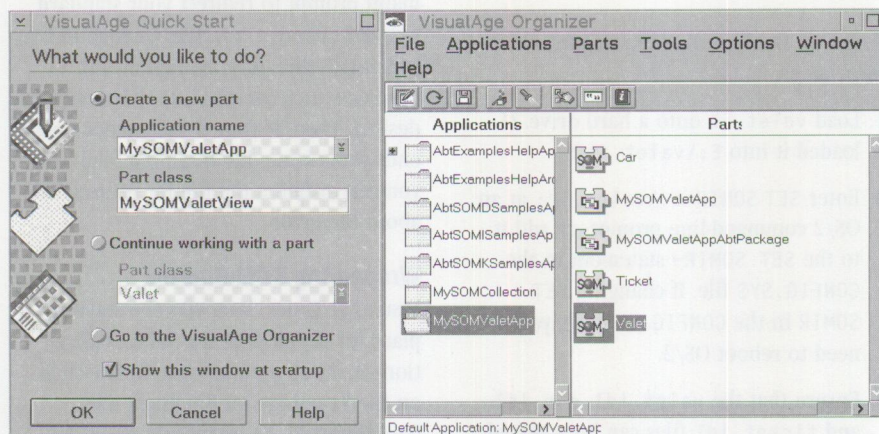


Figure 5. VisualAge Quick Start and Valet SOM Wrappers

'somf_MLinkable is identified in the Car class, so that the Car class can be contained in a collection. This requires that you wrapper 'somf_MLinkable prior to wrapping the VPS classes. To do this:

1. Create an application to wrapper the somf_MLinkable class and name it, for example, MySOMCollection.
2. Next, select your application and open the Create SOM Wrappers window.
3. The Create SOM Wrappers window will list many classes. Scroll to the somf_MLinkable class and select it by clicking mouse button 1 (usually the left mouse button).
4. Press Enter to generate a wrapper for that class.

To more effectively manage the VPS application, you should wrapper the valet parking system SOM classes in a different application. Again, at the VisualAge Quick Start window, enter the name of the application (for example, MySOMValetApp). Then do the following:

1. Select your application and open the Create SOM Wrappers window. The valet application uses Car, Ticket, and

Valet, which can be generated separately or as a group.

2. Select "Car" by clicking mouse button 1.
3. Scroll to "Ticket," then press and hold Ctrl + mouse button 1.
4. Repeat step 3 for Valet.
5. Press Enter to generate the wrappers.

By double-clicking mouse button 1 on one of the generated parts, you'll activate the part Script Editor, which shows the details of the generated wrapper, as Figure 4 illustrates for the Valet part. Note how the SOMsupport wrapping process adjusted the method names to Smalltalk syntax for getter and setter methods (e.g., valetId and valetId:).

Because Car includes somf_MLinkable, SOMsupport will automatically list MySOMCollection as a prerequisite for MySOMValetApp. If you don't wrapper somf_MLinkable prior to wrapping Car, an error occurs in wrapping the Car class.

Because SOM wrapper classes must fit into an existing class hierarchy within VAST, class naming conflicts might occur. You can resolve these conflicts by renaming the SOM wrapper class or method; however, because of the way that the SOM Smalltalk Constructor maps IDL identifiers to Smalltalk identifiers, method names that do not conflict in SOM might conflict in Smalltalk. When method names conflict, the first method is overwritten by the second. Refer to the VAST *Features Class Reference Guide* (on the CD-ROM) for details on avoiding this situation.

Building the UserView

The next step in the process is to use the VAST Composition Editor to build the valet user interface and connect it to the underlying SOM parts. This is one of VisualAge for Smalltalk's significant features—the ability to interactively develop objects that are visual (displayed on a user's screen) and link them to non-visual parts to provide the underlying application logic.

The VisualAge Quick Start window in Figure 5 contains the application name, MySOMValetApp and the Part class MySOMValetView. It also shows the SOM

parts generated for Car, Ticket, and Valet in the VisualAge Organizer window, which we will use from the Composition Editor window. Click on OK to continue.

The VAST Composition Editor provides a tool bar under the menu bar and a parts palette (containing various categories of pre-built parts) on the left side of the screen. A window part serves as the default option in the free-form space. You can select visual parts from the parts palette and drop them in the window part, but non-visual parts are restricted to the free-form surface (see Figure 6).

Your next task is to lay out the screen format. You can either lay it out on paper and then transfer it to the VAST Composition Editor window, or you can interactively lay it out by using the categories, parts, and tools provided with VAST.

The valet parking system application has a basic user interface implementation. Use the following steps to create it:

1. With the cursor over the Window part, press and hold Alt while clicking mouse button 1 to edit the title, making it "Valet Parking System."
2. Select the Data Entry category and the Label part from the Parts palette to add the label for Car.
3. Hold down Ctrl, move the cursor over the label, click mouse button 2 (usually the right mouse button) and drag the label to create label subheadings of Color, Make, and Model, plus labels for Ticket and Valet.
4. Select the Text part (for data entry) and add one next to Color, Make, Model, Ticket, and Valet.
5. Alter the names on the labels by pressing Alt and clicking mouse button 1 at each label.
6. Using the tool bar's alignment tools, align the labels, subheadings, and text fields both vertically and horizontally.
7. Double-click mouse button 1 on the Color text field, select String as its type, and click mouse button 1 twice on the page forward (>) symbol to change the name to Color. Click on OK.
8. Repeat step 7 for the Make, Model, and Valet text fields.

9. For Ticket, select "Integer" as its type, and at page 2, select "right-justify" before changing its name. Click on OK.

Buttons

10. Switch to the Buttons category to add four pushbuttons to the bottom of the Window part. Change the names on the buttons by using the Alt + mouse button 1 method described in step 5.
11. Use the tools on the tool bar to align the buttons horizontally and vertically, then distribute them horizontally, and make them the same size.
12. Click mouse button 2 on each pushbutton to select Open Settings and set the mnemonic as its first letter (this will cause each of the first letters to be underlined).
13. Make "Park Car" the default pushbutton when you adjust its settings.

To summarize the process of adding a part: Select the category, then select the part from the parts palette, and place the part somewhere in the free-form surface or in another part. Figure 7 shows the result of these steps.

Adding SOM Parts

Add the SOM wrappers to the free-form surface by doing the following:

1. From the Options menu, select "Add Part . . ."
2. Type Car in the "Class name" field of the displayed window.
3. Click on OK.
4. Click mouse button 1 on the free-form surface.
5. Repeat for Ticket and Valet.

Figure 7 shows the Composition Editor for MySOMValetView at this point.

Connections

Now add the connections between the parts. To see a part's connections, move the cursor to the part and click mouse button 2. Then select "Connect . . ." with mouse button 1. Figure 8 shows the connection options for Car. The somFree actions are the first connections made, since they ensure that a SOM Car, Ticket, or Valet object created when you test the part will be destroyed and the memory released when it's no longer needed. To add connections:

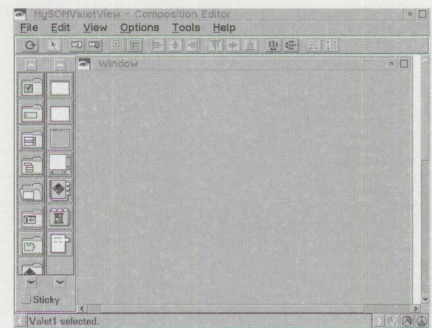


Figure 6. VisualAge Composition Editor for MySOMValetView

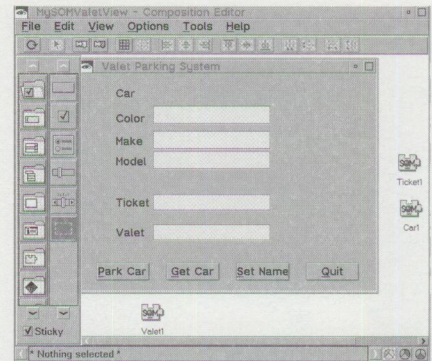


Figure 7. Valet Parking System with SOM Wrappers

1. Double-click mouse button 1 on somFree for Car.
2. Move the cursor over the Valet Parking System title bar and click mouse button 2.
3. Click mouse button 1 on the Window's closedWindow event.

Repeat these steps for the Ticket and Valet SOM parts. Next, you'll add the attribute connections.

4. Press and hold Alt while clicking mouse button 2 on the Color text field to display the field attributes.
5. Click mouse button 1 on object.
6. Move the cursor over "Car" and click mouse button 1 to show all available connections (see Figure 8).
7. Click mouse button 1 on the color attribute to create an attribute-to-attribute connection, shown as a blue line between the Color text field and the Car SOM part.

Repeat these steps for the Make and Model text fields. For the Ticket and Valet

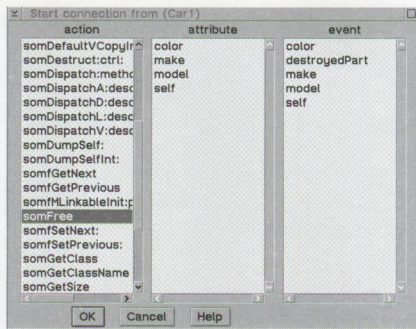


Figure 8. Available Connections for the SOM Car Wrapper

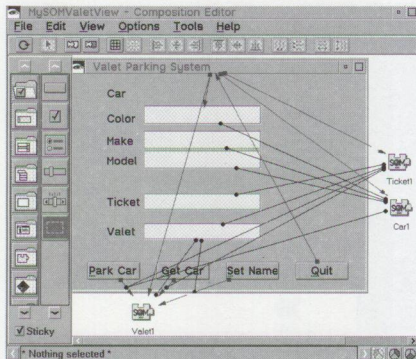


Figure 9. Connections Between VPS VisualAge Parts

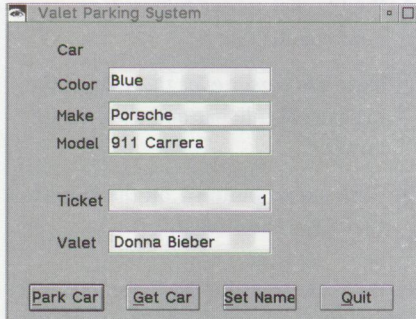


Figure 10. Valet Parking System User Interface

text fields, connect the appropriate attributes to their respective SOM parts.

1. Connect the VPS title event `openedWidget` to the Color actions `setFocus` and `selectAll`. These connections result in the cursor being positioned in the Color text field with any data in that field selected when the VPS application is started.

Now connect the pushbuttons. The first pushbutton to connect is `Quit`, so you can test what you've built thus far.

2. Select the `Quit` "clicked" event using `Alt + mouse button 1`.

3. Move the mouse to the VPS title bar and select the `closeWidget` action, which appears as a green line.
4. Click mouse button 1 on the tool bar's Test icon to validate your user interface layout and ensure that the `Quit` button works.

Connect the `Park Car` button "clicked" event to the Valet SOM part `parkCar`: action, which results in a dotted green line indicating that an object needs to be added as the parameter on this method invocation. The missing object is the `Car` that is to be parked. The `Car` doesn't actually exist in the VAST, but a reference to it is available as "self" of the `Car` SOM part. Connecting `Car` "self" to the line between `Park Car` and `Valet Car` results in a blue line, indicating an attribute-to-attribute connection.

When making this connection, notice the "result" attribute on this "Park Car - Valet" line. Now, connect the "result" attribute between "Park Car - Valet" and the `Ticket` SOM part "self" attribute, since `parkCar`: returns a `Ticket`. Again, test the function to ensure it works.

Connect the `Get Car` pushbutton to the `retrieveCar`: method on `Valet`, similar to the description above, with its result, a `Car` object, connected back to the `Car` SOM part. Also connect the `Set Name` pushbutton to the `Valet` `valetName`. Figure 9 shows the results of these connections.

Test your application by using the tool bar's Test button. Figure 10 shows the VPS's user interface.

Once you've completed development, close the `MySOMValetView` Composition Editor and save the application. You might also save the image at this time to prevent any corruption or loss of your work.

Packaging the Application

Now that you've developed the application, you package it.

1. Select "Smalltalk tools ->Manage Applications" to open the Application Manager window.
2. Scroll to "MySOMValetApp" and click mouse button 1 to select it. Then, under the Applications pull-down menu, select "Package . . ." and choose

"Yes" from the Package dialog box. The "Packaging Applications MySOMValetApp . . ." window appears and displays messages associated with the packaging process.

3. In the Selection Required dialog box, double-click mouse button 1 on "MySOMValetView."

Robust Solutions with Maximum Return

SOM provides significant benefits for enabling applications developed in one object-oriented language to interact with those written in a different language or a language from a different vendor. SOM also offers significant maintenance advantages when new versions of one of those applications are distributed, helping to minimize the effort to keep versions synchronized.

IBM's VisualAge for Smalltalk, a Smalltalk-based tool that provides support for SOM, can help your company implement object-oriented solutions by maximizing your ability to reuse existing code. VAST provides usage bindings and, through the SOMsupport feature, provides a simplified environment for developing SOM-enabled applications that can connect to C, C++, or COBOL SOM applications. IBM is developing implementation bindings for Smalltalk to allow applications written in other languages to reuse Smalltalk applications.



Tom New is an information technology architect within the IBM Personal Systems Competency Center in Roanoke, Texas. He is the advanced object technology team lead and has

worked on distributed systems solutions as an application designer, developer, systems programmer, systems analyst, and lead architect. He was also a systems engineering and an applications development solutions manager, in addition to an IT architect, for the IBM Consulting Group, as well as a national technical support representative. Tom started in object-oriented technology in 1989 as a member of the Common User Access Architecture Review Board. He joined IBM in 1977 after graduating from Cornell University. Tom's Internet ID is tnew@vnet.ibm.com.

SOM Language Neutrality: An OO COBOL Perspective

This article describes an object-oriented COBOL program using SOM-enabled classes written in C++. It illustrates the language neutrality of SOMobjects and introduces many of COBOL's new object-oriented language extensions.

The introduction of object-oriented (OO) language extensions to the syntax of the COBOL programming language is an exciting development in object-oriented programming and design. With object-oriented COBOL, enterprises can adopt a phased approach to object technology and migrate to this new technology in a staged, orderly fashion.

Programmers with a sound foundation in COBOL can write object-oriented programs with minimal training. Indeed, the greatest obstacle programmers face is not learning new verbs and data types, but acquiring an OO mindset.

In the new IBM family of COBOL products, syntax is compatible across workstation, midrange, and host platforms. SOMobjects are also portable across these platforms, meaning that you can develop a COBOL program using SOM on a workstation and port it unchanged to a host or midrange environment.

Robert A. Pittman, Jr.
IBM Corporation
Dallas, Texas

SOM also adds the benefit of language independence; a SOM-enabled class can be developed in SOM's IDL (interface definition language), C++, Smalltalk,¹ or COBOL. And SOM-enabled

¹ Smalltalk supports only usage bindings, not implementation bindings. Hence, classes cannot be invoked from outside the Smalltalk environment, and Smalltalk classes must be exported before they are usable by other languages.

classes can be used by programs coded in different languages. This assertion provides the basis for the program in this article.

OO COBOL Background

The language extensions introduced in IBM's family of COBOL products are based upon the proposed ANSI 1997 standards. Although a number of issues regarding these proposed standards (such as object persistence and exception) have yet to be resolved, there is a consensus about the proposed OO language extensions. This consensus is the basis of the new genre of object-oriented COBOL compilers introduced by IBM and others. As more of the



```

REPOSITORY.
CLASS Car IS "Car"
CLASS Ticket IS "Ticket"
CLASS Valet IS "Valet".

```

Figure 1. Configuration Section Repository

```

*****
*   Define the object handles.   *
*****
01 carObj  USAGE OBJECT REFERENCE Car.
01 ticketObj USAGE OBJECT REFERENCE Ticket.
01 valetObj USAGE OBJECT REFERENCE Valet.

```

Figure 2. Object Handles

proposed standards achieve consensus, compiler vendors will incorporate these standards into their products.

SOM and the IBM COBOL Family

IBM supports the Object Management Group's Common Object Request Broker Architecture (CORBA) for object interoperability. Backing up that support is IBM's SOM, an implementation of CORBA making objects portable across development languages. Both the OS/2 and MVS COBOL compilers feature Direct-to-SOM technology, so you can create language-neutral objects without extensive object-oriented programming experience.

By utilizing a compiler option, you can generate SOM IDL directly from a class definition coded in COBOL. You can then compile this IDL using the SOM compiler, which places it into a SOM interface repository (IR). This process is easy and requires no knowledge of IDL coding.

Development Environment

The COBOL code in this article was written using VisualAge for COBOL for OS/2 with the latest Corrective Service Diskette (CSD) installed on a system running OS/2 Warp. VisualAge for COBOL for OS/2 includes excellent visual programming tools, an interactive debugger, and advanced data utility functions. It also uses the WorkFrame program management tool.

Relevant compiler options used were TYPECHK, which performs typechecking of parameters against method interfaces, and

PGMNAME(MIXED), which allows the use of mixed-case program, class, and method names.

The development machine was a personal computer with a 90 MHz Intel Pentium processor and 32 MB of RAM. For a full installation, VisualAge for COBOL for OS/2 requires 170 MB of disk space, a 486-class processor, and 24 MB of RAM.

Readers familiar with traditional COBOL programming should notice some differences in the code presented in this article.

Class Overview

The sample program in this article uses the valet parking application (see Rick Weaver's "IBM System Object Model—The Wave of the Future [and Now!]" article earlier in this issue). This application consists of Valet, Ticket, and Car classes. Other classes (Garage and TktBooks) are used "under the covers" by the aforementioned classes.

Note: This article's figures show just the specific lines of code being discussed. The complete program is available on the World-Wide Web at <http://pscc.dfw.ibm.com/psmag/>.

Application Summary

As described in Weaver's article, this application reads an input file that instructs the program to either retrieve a car from the garage or park a car in it.

To park a car, you instantiate a car object and set its attributes to those specified on the input record. Next, pass the car object to the parkCar method of the valet object, which returns a ticket object. You can query this ticket object to determine in which slot the car was parked, then display an appropriate message to the system user.

To retrieve a car, you create a ticket object and set its slot number attribute to the value on the input record. Next, invoke the retrieveCar method of the valet object, passing it the ticket object just created. The retrieveCar method returns a car object (or an invalid object if there is no car in the slot). Then obtain the attributes of the returned car object and write a message to the system user.

Implementing in OO COBOL

Readers familiar with traditional COBOL programming should notice some differences in the code presented in this article. These differences arise due to the proposed ANSI 1997 standards on object-oriented extensions to the COBOL language.

One of the first things you will code in this exercise is the configuration section repository. The repository, shown in Figure 1, identifies the classes that the program will need from the interface repository. The client program includes the Car, Ticket, and Valet classes. The subject of a class statement indicates the name of the class as it will be referenced within the program. For example, the Car class in the repository will be referenced as Car in the program.

In the program's working-storage section, object references (a data type introduced in the proposed ANSI 1997 standard) specify the objects' handles and the classes they reference. Essentially, object references are pointers to the area of memory that has been allocated to an object.

Figure 2 lists the object handles defined in the working-storage section. Notice how they relate to the classes defined in the repository.

Notice in Figure 3 that pointers are coded for each text attribute within the objects. These are needed to satisfy typechecking at compilation, when the parameters passed to methods are verified to be compatible with the methods' interfaces.

The valet, ticket, and car objects' attributes are strings. To set one of these attributes, the attribute's value is modified in working-storage, and the appropriate pointer's values are set to the attribute's address. The pointer is then passed as a parameter to the `_set` method. When retrieving an attribute value, the `_get` method returns a pointer, the handling of which will be discussed later.

Figure 3 represents a definition of a string attribute in the working-storage section, with the associated pointer for the string. COMP-5 denotes a binary data item, in this case a halfword in length, and indicates that the native binary format of the environment platform is to represent internal data. Alternately, COMP means that the little-endian format is applied in the PC architecture, while the big-endian format is used on the AIX and MVS platforms.

In the procedure division in Figure 4, a call is made to `somGetGlobalEnvironment`, which returns a pointer to SOM's global environment variable. This pointer must be passed to the methods of the objects used, except when instantiating or destroying an object.

To create an object, invoke the `somNew` method (a new verb—introduced in the ANSI 1997 standard—which is inherited from `SOMObjects`) on the class name of the object specified in the repository. This method returns the object handle and allocates memory for it.

In Figure 5, note the use of `somNew` on the `Valet` class, which returns a valet object. Refer to the repository (Figure 1) and the object reference (Figure 2) in the working-storage section, and observe their relationships. Subsequently, when you invoke the object, you invoke methods on its handle and not its class name.

Notice in Figure 6 that you pass the pointer to the global environment variable as the first parameter when invoking methods on the objects. This parameter, as well as all others used for this exercise,

```
*****
*   Work areas for the valet object's attributes.   *
*****
01  WS-VN-POINTER  USAGE POINTER.
01  WS-VALET-NAME.
    05  WS-VN-LL      PIC S9(4)  COMP-5.
    05  WS-VN-NAME    PIC X(40).
```

Figure 3. Work Areas for Valet Object's Attributes

```
*****
*   SOM global environment variable.               *
*****
01  WS-ev          USAGE POINTER.
.
.
.
*****
*   Obtain pointer to SOM's global environment variable. *
*****
CALL "somGetGlobalEnvironment"  RETURNING WS-ev.
```

Figure 4. SOM Global Environment Variable

```
*****
*   Create a valet object.                         *
*****
INVOKE Valet  "somNew"  RETURNING valetObj.
```

Figure 5. Creating an Object

```
*****
*   Initialize fields and work areas. Valet name is *
*   padded with low values for C++'s null character for *
*   string termination.                               *
*****
MOVE Z"Fritz von Hochsmeier" TO WS-VN-NAME.
SET WS-VN-POINTER TO ADDRESS OF WS-VN-NAME.
*****
*   Set the valet object's name attribute, pass a *
*   pointer to the name work area.                 *
*****
INVOKE valetObj "_set_valetName"
                               USING BY VALUE      WS-ev
                               WS-VN-POINTER.
```

Figure 6. Valet Object's Name Attribute

is passed "by value" rather than "by reference" (the default). *By value* eliminates a level of indirection and is expected by the classes you are using, whereas *by reference* passes a pointer to the parameter,

which is typically used in COBOL programming.

For compatibility, string attributes are null-terminated by moving a low-value (binary zero, X"00") to the position

```

*****
*   Work areas for car object's color attribute.   *
*****
01 WS-CC-POINTER      USAGE POINTER.
01 WS-CAR-COLOR.
   05 WS-CC-LL          PIC 9(4)    COMP-5.
   05 WS-COLOR          PIC X(20).
   05 WS-COLOR-R        REDEFINES WS-COLOR.
       10 WS-CCR-BYTE   PIC X
           OCCURS 20
           INDEXED BY WS-CCR-INDEX.

```

Figure 7. Car Object's Color Attribute

```

LINKAGE SECTION.
*****
*   Define a general work area.   *
*****
01 LS-WORK-AREA      PIC X(100).

```

Figure 8. Obtaining the Value of a String Attribute

```

*****
*   Get the color attribute of the car.   *
*****
INVOKE carObj "_get_color"
           USING BY VALUE   WS-ev
           RETURNING       WS-CC-POINTER
PERFORM GET-COLOR

```

Figure 9. Obtaining the Car Object's Color Attribute

```

GET-COLOR.
*****
*   Gets the color work area from the value pointed to by *
*   the pointer returned by the car object's get color   *
*   method.                                               *
*****
SET ADDRESS OF LS-WORK-AREA TO WS-CC-POINTER.
MOVE LS-WORK-AREA          TO WS-COLOR.
PERFORM VARYING WS-CCR-INDEX
           FROM 1 BY 1
           UNTIL WS-CCR-BYTE (WS-CCR-INDEX) < SPACE
                OR WS-CCR-INDEX > 20
           SET WS-CC-LL TO WS-CCR-INDEX
END-PERFORM.

```

Figure 10. Getting the Color Work Area

```

*****
*   See if a car was returned.   *
*****
CALL "somIsObj"      USING BY VALUE carObj
                   RETURNING theValidFlag

```

Figure 11. Determining Whether a Car was Returned

```

*****
*   The valet clocks out and closes the garage.   *
*****
INVOKE valetObj "somFree".

```

Figure 12. Destroying an Object

immediately following the last character. Figure 6 sets the valet object's name attribute. The prefix Z before the literal indicates that it will be null-terminated. Figure 6 also sets up a pointer to the attribute's address, to be passed to the method. The object's handle is invoked, rather than the class name, which was invoked during object creation.

Because the `_get` methods return pointers to the attributes they are getting, and because COBOL does not allow altering the address of an item defined in working-storage, I coded a variable in the linkage section that is used as a work area. The address of this variable is set to the pointer value returned by the `_get` method. The contents are then copied to the attribute defined in working-storage, and the length up to the null-termination character is determined. For simplicity's sake, I arbitrarily assigned a length of 100 bytes to the variable, allowing me to use it for all string attributes referenced in the program.

It may seem odd to set up a linkage section in a program that is executed directly and never called, but it is necessary to obtain the values of the objects' string attributes. Figure 7 gives the working-storage definition of the car object's color attribute. Figure 8 shows the generic work field to be used to obtain the value of a string attribute. Figure 9 determines the car object's color attribute. Notice how a pointer is returned.

The code in Figure 10 uses the returned pointer to address the variable defined in the program's linkage section and copies the attribute to the working-storage variable. The length of the attribute up to its null-termination character is also found.

General agreement on exception handling has not yet been reached in the proposed ANSI 1997 standards, so the C++ techniques of "throwing" and "catching" exceptions can not be emulated. To verify that a valid car object was returned by the valet object's `retrieveCar` method, you call `somIsObj`, passing the handle of the car object. `somIsObj` returns a flag indicating whether the object is valid or invalid.

In Figure 11, `somIsObj` is called to see if an object returned by another method is valid. `ValidFlag` is subsequently tested

to determine the outcome of the call to somIsObj.

To destroy an object, use somFree, a method inherited from SOMObject, as shown in Figure 12. It frees the memory allocated to the object, because (as in C++) there is no automatic garbage collection in object-oriented COBOL. somFree is invoked on the object's handle, rather than the class name Valet.

Developing the Client Program

To develop the client program with VisualAge for COBOL for OS/2, first create .LIB files for the DLLs supplied for the classes used by executing the following command from an OS/2 command line:

```
IMPLIB valet.lib valet.dll
```

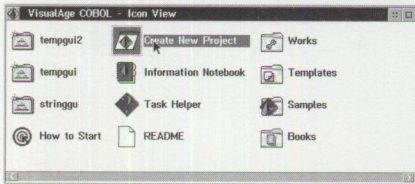


Figure 13. VisualAge COBOL—Icon View

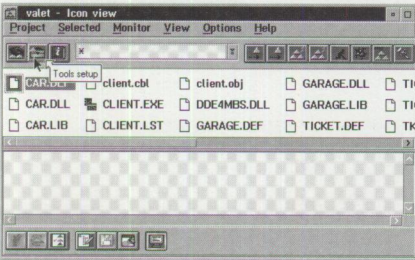


Figure 14. Valet—Icon View

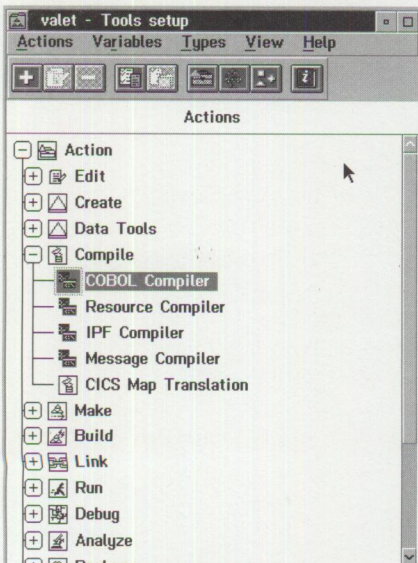


Figure 15. Valet—Tools Setup

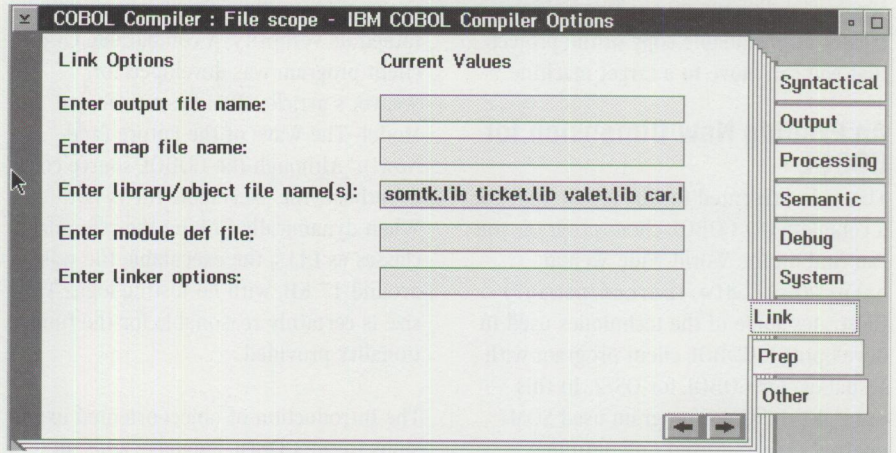


Figure 16. Compiler Options

Next, open up the VisualAge COBOL Icon View and select the "Create New Project" icon (Figure 13).

From the "Create New Project" window, select "Create a default COBOL project" with a project name of valet and a target name of client.exe. Next, select the "Create New Text File with the COBOL Editor" button on the toolbar and enter the source code for the client program.

Then copy the VALET.IR file, all the .DLL and .LIB files, and the test file CAR.SCR into the valet project's subdirectory. You need the .LIB files to build the project and the .DLLs and test file for execution. You also need the VALET.IR file to satisfy class typechecking during compilation.

Next, open the "Tools Setup" option from the toolbar (Figure 14). From here, select "COBOL Compiler" from the Actions list (Figure 15), which presents a notebook of compiler options (Figure 16).

In this notebook, select the "Link" tab and enter the names of the TICKET.LIB, VALET.LIB, and CAR.LIB files in the library/object file name(s) entry box. You also need the SOMTK.LIB file; this file is supplied with VisualAge for COBOL for OS/2. Then close the compiler options notebook.

In the "Tools setup" menu, select the variables pull-down menu (Figure 17) and add a CARS variable with an associated string of car.scr, the name of the file containing your test data.

After closing "Tools setup," you build the executable file by selecting one of the "Build" options on the toolbar. When the project is built successfully, execute it by selecting the "Run Target" option on the toolbar (Figure 18).

If you want, you can package the project's executables and requisite runtime files by selecting the "package" option in the

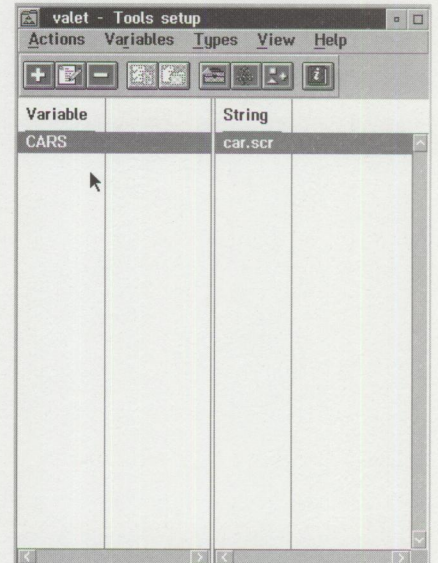


Figure 17. Valet—Tools Setup

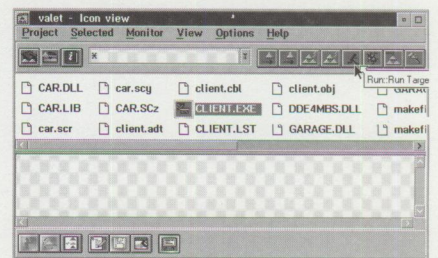


Figure 18. Valet—Icon View

"Project" pull-down menu. This process creates an installable copy of the project that you can move to a target machine.

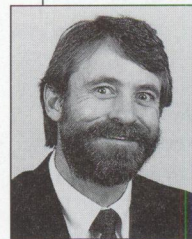
An Exciting New Dimension for COBOL

The code presented in this article (part of a complete OO COBOL client program you can find on the World-Wide Web at <http://pscc.dfw.ibm.com/psmag/>) illustrates some of the techniques used in developing a COBOL client program with VisualAge for COBOL for OS/2. In this example, the client program used SOM-enabled classes. I used only minimal VisualAge COBOL functionality. Coding a class definition in object-oriented COBOL is another exercise beyond the scope of the topic at hand.

When comparing this COBOL client program to a functionally identical C++ client

program, note that COBOL has maintained its verbosity. A comparable C++ client program was developed for Weaver's article, "IBM System Object Model—The Wave of the Future (and Now!)." Although the COBOL source code is verbose, the executable file is not. When dynamically linking the referenced classes as DLLs, the executable file was around 17 KB, with no testing logic. This size is certainly reasonable for the functionality provided.

The introduction of object-oriented extensions to the COBOL language adds an exciting dimension to the world's most widely used programming language. These extensions ensure the survival of its widespread usage, paving its way into the new generation of application development.



Robert A. Pittman, Jr. works in the Object Technology University within IBM Education and Training, Dallas, Texas. He provides instruction and develops courses in object orientation,

and he is a co-author of *IBM VisualAge for COBOL for OS/2: Object-Oriented Programming* (SG24-4606), an IBM Redbook. Rob's prior IBM jobs were in Availability Services. He has been involved in the computer industry since 1976. Before joining IBM in 1988, he worked in application development, database administration, and systems programming. Rob has a BBA degree in Accounting from the University of Texas and an MBA in Information Systems from the University of North Texas, and he is a Certified Public Accountant licensed in Texas. His Internet ID is robert_pittman@vnet.ibm.com.

Corrective Service Information

Figure 1 shows maintenance release levels for the listed products. This information is effective as of February 1, 1996. CSDs may have been updated since press time.

To order all service packages—except for the OS/2 2.0, OS/2 2.1, OS/2 2.1 for Windows, and OS/2 2.0 Toolkit ServicePaks—call IBM Software Solution Services at (800) 992-4777. For the OS/2 2.0 ServicePak (XR06100), OS/2 2.1 ServicePak (XR06200), OS/2 2.1 for Windows ServicePak (XR06300), or the

IBM Developer's Toolkit for OS/2 2.0 ServicePak (XR06110) on diskettes or CD-ROM, call (800) 494-3044. Most OS/2 service packages are also available electronically from the following sources:

- **OS/2 Bulletin Board Service (BBS):** In Software Library, select Option 2. (Corrective services are also listed under the General category on the IBMLink BBS.) To subscribe to the OS/2 BBS, call (800) 547-1283.
- **IBM Personal Computer Company (PCC) BBS:** Call (919) 517-0001.

Service packages are located in Directory 4.

- **CompuServe:** Download service packages from the IBM OS2 FORUM library (GO IBMOS2 IBM DF2).
- **Internet:** Do an anonymous FTP from `ps.boulder.ibm.com` at `/ps/products/`. TCP/IP packages are located at `software.watson.ibm.com` at `pub/tcpip/os2`.

—Arnie Johnson, IBM Corporation, Austin, Texas

Product/Component	Release	CSD Level	PTF Number	Change Date	Comments
OS/2 Standard Edition	1.3	XR05150	XR05150	02-10-93	
OS/2 Extended Edition	1.3	WR05200	WR05200	05-12-93	WR05200 replaces WR05050, which can no longer be ordered on diskette
OS/2	2.0	XR06100	XR06100	09-01-93	XR06100 replaces XR06055.
OS/2 2.10 ServicePak	2.1	XR06200	XR06200	03-01-94	This package is not for OS/2 2.1 for Windows.
OS/2 2.11 for Windows ServicePak	2.11	XR06300	XR06300	05-24-94	
OS/2 Toolkit	2.0	XR06110	XR06110	09-01-93	
	1.3	XR05053	XR05053	03-23-92	
OS/2 LAN Server/Requester ServicePak	2.0	IP06030	IP06030	04-25-93	
OS/2 LAN Server/Requester ServicePak	3.0	IP07060	IP07060	05-10-95	Supersedes IP07045.
IBM LAN Server/Requester OS/2 Warp Connect LS 4.0 ServicePak	4.00	IP08152	IP08152	11-28-95	Supersedes IP08150.
OS/2 Extended Services Database Manager ServicePak	1.0	WR06035	WR06035	11-18-93	Supersedes WR06001, WR06002, WR06003, WR06004, WR06014, and WR06015.
DB2/2 ServicePak	1.0	WR07042	WR07042	06-08-95	
DB2/2 ServicePak	2.1	WR08049	WR08049	10-12-95	
DDCS/2 ServicePak	2.0	WR07031	WR07031	02-06-95	
Database Manager DB2/2	1.2	WR07047	WR07047	06-06-95	
DDCS/2	2.0	WR07046	WR07046	06-06-95	
Client Application Enabler/2 (CAE/2)	1.2	WR07043	WR07043	06-06-95	
Software Developers Kit/2 (SDK/2)	1.2	WR07048	WR07048	06-06-95	
SDK/Windows ServicePak	2.1	WR08050	WR08050	10-12-95	
Extended Services Comm Mgr ServicePak	1.0	WR06025	WR06025	11-29-93	

Figure 1. Maintenance Release Levels (continued on next page)

Product/Component	Release	CSD Level	PTF Number	Change Date	Comments
System Performance Monitor (SPM/2) ServicePak	2.0	WR06075	WR06075	12-10-93	
LAN Distance ServicePak	1.1/1.11	IP08175	IP08175	12-21-95	Supersedes IP07050.
OS/2 Network Transport Services/2 SelectPak	2.11/2.20.1 2.20.2	WR07060	WR07060	05-10-95	Must be LAPS 2.11 or above. If not, order WR07045 first.
LAN Server 4.0 MPTS	4.0	WR08150	WR08150	10-18-95	
LS 4.0 MPTS Warp Connect	1.0	WR08152	WR08152	11-06-95	
Communications Manager/2 Version 1.01 ServicePak	1.01	WR06050	WR06050	06-11-93	Available only on diskette.
CM/2 Version 1.11 ServicePak	1.11	WR06150	WR06150	05-31-94	Available on diskette and CD-ROM.
DOS	4.0/4.01	UR35284	UR35284	09-26-91	
	5.0	UR37387	UR37387	09-22-92	
C Set/2 Compiler	1.0	CS00050	XR06150	06-29-93	
C Set C++ Compiler	2.0/2.01	CTC0002	XR06102	12-15-93	
C Set C++ Compiler	2.0/2.01	CTC0010	XR06190	09-15-94	
C Set C++ Utilities	2.01	CTM0006	XR06196	09-15-94	
C Set C++ Utilities	2.00	CTL0007	XR06197	09-15-94	
TCP/IP for OS/2 Base and Application Kit	2.0	UN64092	UN64092	08-24-94	
TCP/IP for OS/2 DOS Access	2.0	UN57546	UN57546	08-24-94	
TCP/IP for OS/2 Extended Networking	2.0	UN60005	UN60005	06-21-94	
TCP/IP for OS/2 Programmer's Toolkit	2.0	UN57887	UN57887	06-21-94	
TCP/IP for OS/2 Domain Name Server	2.0	UN60004	UN60004	08-24-94	
TCP/IP for OS/2 Network File System	2.0	UN57064	UN57064	06-21-94	
TCP/IP for OS/2 X-Windows Server	2.0	UN68122	UN68122	01-20-95	
TCP/IP for OS/2 X-Windows Client	2.0	UN59374	UN59374	08-24-94	

Figure 1. Maintenance Release Levels

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

Advanced Peer-to-Peer Networking, AIX, AIX/6000, APPN, Aptiva, AS/400, BookManager, BookMaster, Common User Access, Communications Manager, C Set ++, CUA, DATABASE 2, DATABASE 2 OS/400, DB2, DB2/2, DB2/400, DB2/6000, Distributed Database Connection Services/2, DProp, DRDA, DSOM, DualStor, IBM, IBMLink, IIN, LAN Distance, LANStreamer, Micro Channel, MVS, MVS/OE, NetFinity, NetView, OS/2, OS/400, Person to Person, PowerPC, Presentation Manager, PS/2, RISC System/6000, ServicePak, SOM, SOMobjects, System/390, TalkLink, ThinkPad, UltimeMedia, ValuePoint, VisualAge, VisualGen, VM, VoiceType, WebExplorer, WIN-OS2, Workplace Shell, XGA

Windows is a trademark of Microsoft Corporation.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

These back issues of *Personal Systems* are available to provide valuable information. Indicate the desired quantity for the issues you want to order and complete the information on the following page.

January/February 1996

What's New?
 Tape Backup Products for OS/2
 Fault Tolerance for LAN Server
 Getting Together with cc:Mail
 Sales Force Automation: Building the Intelligence-Driven Sales Organization
 The New Mercantilism
 Designing Lotus Notes Applications That Perform
 Designing a Scalable Lotus Notes Workflow Application
 Lotus Notes for AIX in a Personal Systems Environment
 New Administrative Features and Enhancements in Lotus Notes Release 4
 MQSeries link for Lotus Notes
 Getting Warped and Connected Too!—Part Two

November/December 1995

What's New?
 Road Trip! Shopping the Internet
 Command-Line Commando
 Getting Warped and Connected Too!
 Infrared: LANs Without Wires
 Security and Auditing in IBM LAN Server
 Multi-User Performance Testing in a Client/Server Environment
 DCE Cell Performance: High Water Marks
 Plug and Play in PC DOS 7

September/October 1995

What's New for OS/2?
 Mesa 2 for OS/2
 Manage Your Files with FileStar/2 for OS/2
 PartitionMagic for OS/2
 Managing LAN Server Home Directories
 IBM DualStor for OS/2
 Human-Computer Interaction Overview
 User Interface 2000
 IBM's Strategy for OS/2 Platform Products Fix Support
 Road Trip! Back to School
 TalkLink Gets a Facelift
 OpenDoc and Human-Computer Interaction
 Supporting HCI Technologies in Applications
 An Introduction to Speech Recognition with OS/2
 Intelligent Agents: A Primer
 CID Installation of OS/2 and Its Platform Applications
 Creating Your Own INF Hyperlinked Files

July/ August 1995

What's New for OS/2?
 The Soap Box Derby
 Easily Load and Lock Desktops
 Road Trip! Cruisin' to the Olympics
 DB2 for OS/2 V2.1: The Next Generation
 OS/2 Victories from the Data Management Front Lines
 Voting Kiosks: The Future of Electronic Elections
 Performance Enhancements in DB2 for OS/2 V2.1
 DB2 for OS/2 Administrative Tools
 Database Recovery with DB2 for OS/2
 Getting Object-Oriented with DB2 for OS/2 V2.1
 Enhanced SQL in DB2 for OS/2 V2.1
 Enterprisewide Connectivity Using DB2
 Visualizer Development
 Performance: DCE RPC as a DB2 for OS/2 and DB2 for AIX Transport
 Remote Program Load of OS/2 Warp from NetWare 3.12

May/June 1995

What's New for OS/2?
 Thanks for the Memory
 Road Trip! Disney on the Internet
 Apache Students Use the Power of the Pen (Light Pen)
 Visualizer: The Conversion Continues
 The Internet: A New Dimension?
 IBM LAN Doctor Services
 Borland C++ 2.0 Brings OWL to the OS/2 Presentation Manager
 LAN Server Logon Internals
 LAN Server 4.0 Performance, Capacity Enhancements, and Tuning Tips
 OS/2 Warp for Developing PC Games
 Controlling the OS/2 Desktop From a File Server
 Jump-Start Your PC with Component Upgrades

March/April 1995

What's New for OS/2?
 Mesa 2: Gaining the Competitive Edge with OS/2
 Managing the Workplace Shell with DeskMan/2
 Circus du COMDEX: The Running of the Geeks
 Road Trip! Touring the Side Roads of the Internet
 What's New in PC DOS 7
 OS/2 Boot and Recovery Options
 TCP/IP: How It Works
 A Guide to OS/2 Warp's Internet Access Kit
 CID Installation of OS/2 Warp and LAPS
 Wrapping Up an OO Experience

January/February 1995

Technical Connection Personal Software Is the Answer!
 Visualizer, DB2, and You—An End-User's Perspective
 Insiders' Software Unveiled
 Need a Specialist for Your LAN Server 4.0?
 One-Stop Shopping
 OS/2 Warp
 OS/2 for SMP
 Multimedia File I/O Services
 Need a Fix?
 IBM LAN Server 4.0: New Features and Comparisons with NetWare
 IBM DCE Heterogeneous Enterprise Performance

November/December 1994

Evolution, Not Revolution—Pen Computing Comes of Age
 Handwriting Recognition: The State of the Art
 Pen Digitizing Hardware
 It's HapPENing!
 Bill Carr: Fastest Draw in the West
 Work Management in the Field
 Communicating Without Wires: IBM's Mobile Communications Module
 Tomorrow's Networking Today—from IBM's Personal Systems Competency Center
 Customers Speak Out About Consult Line
 New DeScribe 5.0—Leader of the Pack
 Super-Fast PenDOS
 Pen for OS/2
 A Development Environment for Pen-Centric Applications
 Writing DOS Installation Programs for Selective Boot Systems
 OS/2 for PowerPC: Transforming Architecture into Implementation

September/October 1994

"Sneaker Net" or Systems Management?
Like Father, Like Son
The Book Shelf
Cajun Electric Cooks Up OS/2 GUI with VisPro/REXX!
Application Development by Program Integration
IBM REXX for NetWare
GammaTech REXX SuperSet/2—Give Your REXX Programs the Power of C
BranchCard: A Viable Option to Stand-Alone Hubs
A Hands-On Primer for REXX
Visual REXX Development Environments
CID Installation of OS/2 2.11 and LAPS
Upgrading from Microsoft LAN Manager to IBM LAN Server 3.0
Stretching Your LAN with LAN Distance
DB2/2—More Than Ever Before!
NetBIOS, SNA, and NetWare IPX Coexistence Under OS/2

July/August 1994

IBM's Personal Systems Support Family—Customer-Influenced Design
OS/2 Times and Scores the 1994 Indianapolis 500

Software Compatibility: Good Relationship or One Night Stand?
Migrating Windows Applications to OS/2: Easing the Migration Path
OS/2 Conference Draws Praise
DCE: An Application Primer
Distributed Performance Characteristics of IBM DCE for OS/2
Architecture Soup: Understanding Modern IBM PC Architecture
TSHELL: A Text-Based Alternate Shell for OS/2
Extended Attributes for Files
Developing Lotus Notes Applications
Conserving Power with Personal System Power Management
Superstor/DS Data Compression in PC DOS 6.x
LAN NetView Object Registration Services

May/June 1994

"Wrightsizing" at USAir
Getting the Word Out at Chemical Banking Corporation
Back Up for the Future
Lost in Cyberspace
The Book Shelf
Threads
Redirected Installation of OS/2 2.x

Send this form with a check or money order, payable to **NCM Enterprise**, to: NCM Enterprise, P.O. Box 165447, Irving, TX 75016-9939. You can also fax both pages of this form to **(214) 518-2507** (please include VISA/MasterCard/AmEx/Diners number and expiration date), or call **(800) 678-8014**. All orders must be prepaid. Checks must be in U.S. dollars.

BACK ISSUE ORDER FORM

NAME _____

COMPANY _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

TELEPHONE (_____) _____

Price is \$12.00 per issue, plus \$3.95 shipping & handling per copy. Overseas orders add \$9.95 shipping & handling per copy.

Texas residents add applicable sales tax.

I have enclosed a: Check Money order
Charge to: VISA MasterCard AmEx Diners

CREDIT CARD NUMBER _____

SIGNATURE _____ EXPIRES _____

IBM believes the statements contained herein are accurate as of the date of publication of this document. However, IBM hereby disclaims all warranties as to materials and workmanship, either expressed or implied, including without limitation any implied warranty of merchantability or fitness for a particular purpose. In no event will IBM be liable to you for any damages, including any lost profits, lost savings, or other incidental or consequential damage arising out of the use or inability to use any information provided through this service even if IBM has been advised of the possibility of such damages, or for any claim by any other party.

Some states do not allow the limitation or exclusion of liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This publication could contain technical inaccuracies or typographical errors. Also, illustrations contained herein may show prototype equipment. Your system configuration may differ slightly.

IBM has tested the programs contained in this publication. However, IBM does not guarantee that the programs contain no errors.

This information is not intended to be a statement of direction or an assertion of future action. IBM expressly reserves the right to change or withdraw current products that may or may not have the same characteristics or codes listed in this publication. Should IBM modify its products in a way that may affect the information contained in this publication, IBM assumes no obligation whatever to inform any user of the modification.

Some of the information in this magazine concerns future products or future releases of products currently commercially available. The description and discussion of IBM's future products, performance, functions, and availability are based upon IBM's current intent and are subject to change.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not imply giving license to these patents.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such

references or information must not be construed to mean that IBM intends to announce such products, programming, or services in your country.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever.

The articles in this publication represent the views of their authors and do not necessarily represent the views of IBM. This publication may contain articles by non-IBM authors. IBM does not endorse any non-IBM products that may be mentioned. Questions should be directed to the authors.

Publication of advertising material in this magazine does not constitute an expressed or implied recommendation of endorsement of IBM of any particular product, service, company, or technology. IBM takes no responsibility whatsoever with regard to the selection, performance, or use of any advertised products. All understandings, agreements, or warranties must take place directly between the vendor and prospective users.

climb, and snorkel. 8-2420
SEEKING successful, intelligent individuals
for spirited annual group encounter. AS/400-
loving, AIX-oriented, OS/2-involved, S/390-
addicted, Internet-obsessed, cross-platform
developers welcome. Five glorious days in
Opryland USA. ☎ 1-800-872-7109.
ORBL WM, 22, very handsome



Come to the 1996 IBM International Technical Interchange, April 22-26 in Nashville, TN. There's a lot to experience: Insights into IBM's latest software strategies. Over 300 unique elective sessions, including App Development, Network-Centric and Client/Server Computing. Tips and techniques for building a competitive advantage. More than 200 exhibitors. Free Developer Connection software. And, of course, some nifty

T-shirts. For enrollment information and program brochure, call 1 800 872-7109 (U.S. & Canada) or 1 617 893-2056 (Outside N. America). Or visit our Web site at http://www.austin.ibm.com/developer/conferences/ti_96 for details.



Solutions for a small planet™

The Two Best Ways to get the most out of OS/2

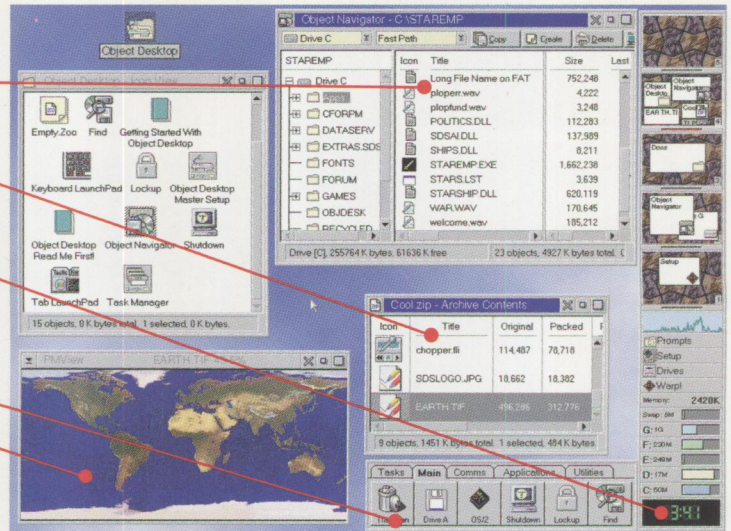
Object Navigator, OS/2's first object oriented file manager can see long file names on FAT partitions, shadows, and program-objects

Object Archives treats ZIP files as standard OS/2 folders

Control Center monitors resources, organizes folders, and provides virtual desktops

Tab LaunchPad organizes programs

View and edit data right from the zip files



Plus!

Hypercache, OS/2's first third party cache.

Browse Mode, enter sub-folders quickly and seamlessly.

HyperDrives, speeds up the opening of folders.

Keyboard LaunchPad, assign hot keys to programs.

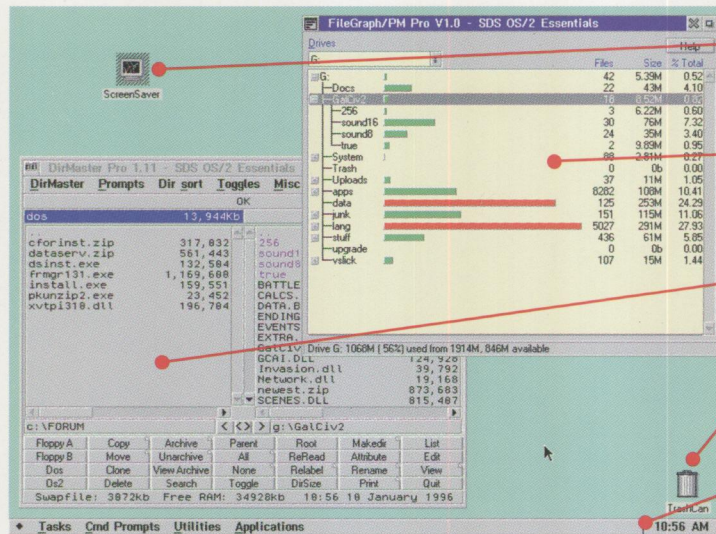
Text View, makes editing and viewing data easier than ever.

Global settings for full drag, sorting, folder views and more.

And a lot more!

Object Desktop is the next generation desktop environment designed exclusively for OS/2. It will make your OS/2 system more powerful, easier to use, and just plain nicer to look at.

Stardock™
**Object
DESKTOP**



Screen Saver Pro secures unattended machines while providing green monitor support.

FileGraph/PM Pro maps out how much disk space each directory is using up on the system.

DirMaster Pro provides one of the fastest and most powerful file managers for OS/2 makes maintaining an OS/2 system a snap.

WPS Trashcan provides the ultimate in deletion protection.

FileBar makes getting to your programs easier while providing a scheduler to launch programs.

OS/2 Stardock
Essentials™

OS/2 Essentials is a powerful yet affordable utility suite for OS/2. It provides a comprehensive of "essential" utilities to complete your OS/2 system. You can now find it almost everywhere for less than \$30.00!

Stardock™

Stardock Systems, Inc.
Phone: 313-453-0328
Email: stardock95@aol.com.
WEB: <http://oeonline.com/~stardock>

Now available at a reseller near you

Micro Center CompUSA Indelible Blue Kiyo Design OS+Resource Cosmos/2

Visit Stardock Systems at the IBM Technical Interchange at Booth #508!

Circle #16 on reader service card.