December 2000, Volume 6, Number 12

Cover Art By: *Arthur Dugoni*

ON THE COVER

I've Got a Secret

Let's create a development environment that combines the ease-of-use of VB with the power of OOP and C++. Let's make it component based, but still able to statically compile for maximum performance and minimal DLL conflicts. We'll have it support Microsoft technologies better and sooner than VB, and we'll even create a Linux version for cross-platform development.

Then let's keep it a secret.

Sound familiar? If you're a long-time Delphi developer, as I am, you recognize this truth-is-stranger-than-fiction scenario all too well. Through several management upheavals, countless premature obituaries by industry pundits, and dizzying swings in business strategy, there's been one constant: pathetic marketing.

We all slept through the "On Time, On Budget" campaign, carefully synchronized with the "Inprise" branding fiasco. Now, as the Inprise emblem has slowly crept into the shadow of the Borland name, we have the "Webvolutionaries" campaign, seemingly inspired by Elmer Fudd. Brilliant.

Borland makes better development tools than Microsoft, or anyone else. They make the best application server out there, but who would know? They are the Mercedes of developer tools, with the marketing strength of a Daewoo. Allaire, Embarcadero, Scour.com, and countless others leverage Delphi as their "secret weapon," producing top-notch software in record time, because they know The Secret. Unfortunately, they're in the minority.

On this late August day, Dice.com is trumpeting some 153,752 open positions. Of those, only 627 are matches for jobs requiring Delphi. That's one in every 245 jobs, or 0.4%. Compare that to 14,950 Visual Basic jobs, approximately one in every 10, or 9.7%. And Java? Java (not JavaScript) weighs in with 29,073 — one in every 5 jobs — or 18.9%! So our admittedly unscientific, but still relevant, search shows that, for every 200 IT jobs across the US, approximately 38 will be Java jobs, 20 will be Visual Basic jobs, and less than one will be a Delphi job.

The persistent lack of exposure of Delphi and other Borland tools has left those who have invested their time and talents in Borland technologies scrambling to diversify. "Borland continues to make great solutions, but to attract desirable business I have added better-marketed solutions from other vendors. Personally, I find myself having to weigh the market presence of competing solutions as much, or more than, the technical aspects. Marketing matters," notes Kyle Cordes, a Delphi and Java consultant and editor of The BDE Alternatives Guide at http://kylecordes.com/bag. Anthony DiAngelus, a senior architect and consultant from Tampa, Florida concurs. "Unfortunately, I've had to rely on my other skills as an Oracle DBA, rather than be a full-time Delphi developer. The work just isn't there, and I can only point the finger at the marketing group. Delphi is the best development tool I have, but corporate America just doesn't know about it."

Is it just a case of Borland being another aim-impaired David against the Goliath of Redmond? Certainly, it's not easy to market against Microsoft's tools division, which is heavily stocked with ex-Borlanders, and has very deep pockets. However, companies like WebGain (http://www.webgain.com) are successfully bundling top-quality tools — and getting noticed. Just try to get through an *InfoWorld* or other trade magazine and not see them. Where is Borland's bundling strategy and marketing presence? Aiming to be Switzerland doesn't get you on many people's maps.

In their defense, Borland has begun, finally, to target the VB community with their latest campaign, featuring the promise of a Linux-based RAD tool in Kylix (http://www.vbforlinux.com) and aggressively distributing functional demonstration versions of Delphi in publications like *Visual Basic Programmer's Journal*. But is it too little, too late?

Kylix may be a reality by the time you read this, but the Linux market is still an emerging one. The Windows market will continue to flourish for many years, and Delphi — the most powerful Windows development environment available — will go the way of FoxPro unless someone turns the lights on at Borland marketing.

— *Michael Pence*

Michael Pence (mikepence@yahoo.com) lives in Phoenix with his wife, Denise, and children, Ryan and Becca, and Sammy the puppy. He's been an analyst and programmer for 10 years, using Java, Delphi, Visual Basic, and other tools. He is also founder of The Delphi Advocacy Group (http://www.egroups.com/group/tdag), a mailing list for Delphi users.

Delphi TOOLS

New Products and Solutions

Opaque Software Announces WithPalette 5 and WithView 5

Opaque Software announced the release of *WithPalette 5* and *WithView 5*, the latest versions of the company's IDE experts for Delphi 4 and 5 and C++Builder 4 and 5.

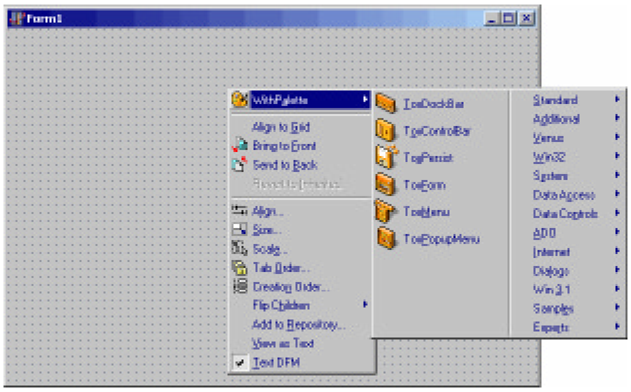
WithPalette is an expert designed to enhance the IDE by making it easier for the developer to place components on forms. WithPalette lets developers instantly add a component to any form with a right-click of the mouse.

WithPalette can be used with or without the IDE's Component palette. WithPalette allows the developer to hide the standard IDE Component palette, providing more screen space, yet allow quick access to all palette components. Also, WithPalette can be configured to give quick access to favorite or most-recently-used components.

WithView is designed to enhance the IDE by making it easier for developers to navigate the various windows and designers. WithView adds a toolbar and a menu containing every open window in the IDE, allowing the developer to select any window with a click of the mouse. The keyboard can also be used to select any window using the WindowSwitcher (similar to pressing Alt Tab).

WithView can automatically "bring to front" the Object Inspector when a form in the IDE is selected. Likewise, the Code Explorer can be brought forward when the Editor window is used. The developer can also make any window in the IDE a Stay-On-Top window or choose to place any window on the Windows Taskbar at the bottom of the screen.

Opaque Software
Price: US$19 each.
Phone: (707) 451-8357
Web Site: http://www.opaquesoftware.com



Global Majic Announces 3DLinX

Global Majic Software, Inc. announced *3DLinX*, a real-time graphic rendering engine and development tool designed to make realistic 3D applications easier to create. 3DLinX uses COM and ActiveX to bring models to life that are aware of each other, interact with one another, obey the laws of physics, and have unique behavior and characteristics.

Developers can command properties, methods, and events of 3DLinX objects. 3DLinX can be used with a developer's favorite programming environment, including Delphi, C++Builder, Web pages, and others. It also provides design-time scene visualization, allowing the programmer to interact with the scene at design time.

3DLinX developers do not need to worry about incorporating complex mathematics, accessing complex graphics APIs, optimizing rendering code, detecting collisions, implementing multi-threading optimizations, exploiting multiple CPUs and hardware acceleration, and applying textures, transparencies, lights, 3D sounds, perspectives, projections, screen overlays, and other complex functions.

The 3DLinX Standard Edition includes loaders for importing models in 3D Studio, Version 3.0 (.3ds); Open Flight, Version 14.2, 15.x (.flt); and Cory-phaeus, Version 3.0-4.2 (.dwb). Additional loaders will be available as add-ons.

3DLinX supports SGI RGB (RGB, RGBA, INT, INTA, BW); BMP; TGA (24- and 32-bit); TIF (uncompressed 8-bit and 24-bit); and JPEG (8-bit and 24-bit).

3DLinX also provides a mechanism for concealing valuable source code, as well as a linking feature.

Add-on products can be added to the 3DLinX architecture, including Living Models; Keyframes and Inverse Kinematics capabilities for animators and game developers; and others.

Global Majic Software, Inc.
Price: US$895
Phone: (877) 3DMAJIC
Web Site: http://www.3dlinx.com

SilverLakeTech.com Announces PC Data Finder

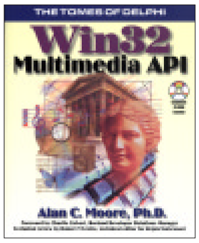
SilverLakeTech.com announced *PC Data Finder*, a PC search engine that enables the user to find anything on a hard drive or removable drive.

PC Data Finder will do searches through ZIPs, PDFs, HTML, files with Office Suite extensions, and many others. PC Data Finder can also do Boolean, proximity, phrase, and wildcard searches on documents in any directory or drive.

PC Data Finder can also search a user's e-mail by Body, Title, Keywords, Subject, Author, To, Date, Header, Footer, or Operator. PC Data Finder works by indexing every word in every document in the directories you choose to index. This enables the user to search for a document based on its content.

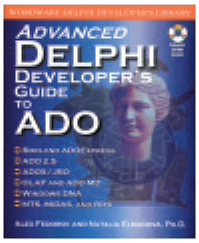
SilverLakeTech.com
Price: US$99.55
Phone: (973) 259-9300
Web Site: http://www.silverlaketech.com

Win32 Multimedia API
Alan C. Moore, Ph.D.
Wordware Publishing, Inc.



ISBN: 1-55622-666-7
Price: US$54.95
Web Site: http://www.ComputerBookstore.com

Advanced Delphi Developer's Guide to ADO
Alex Fedorov and Natalia Elmanova
Wordware Publishing, Inc.



ISBN: 1-55622-758-2
Price: US$59.95
Web Site: http://www.ComputerBookstore.com

Delphi TOOLS

New Products and Solutions

TurboPower Announces Sleuth QA Suite 2

TurboPower Software Co. announced *Sleuth QA Suite 2*, a new version of the company's quality assurance tools suite for professional programmers. With Sleuth QA Suite 2, developers can find bugs, optimize program performance, and test the projects they develop. Sleuth QA Suite 2 includes five tools.

Sleuth CodeWatch tracks down hard-to-find memory and resource leaks; tests for improper use of Microsoft Windows API calls; tests to ensure that API calls used in a program will be available on the developer's target platform; and reports errors to the line of source code where they start.

Sleuth StopWatch times program execution at a macro level and reports exactly how much time is consumed by each routine in a project. Built-in KeyViews provide easy access to commonly required information, such as "The Top 10 Most Time Consuming Routines" and "The Top 25 Most Called Routines."

Sleuth LineProfiler times program execution at the source-line level. Developers can get accurate timing statistics for any or all of the lines of their projects.

Sleuth CoverageAnalyst tracks the number of times individual lines of a project are executed and visually indicates the portions of the project that have not been adequately tested.

Sleuth ActionRecorder records keyboard and mouse interactions for unattended, automated test suites. Recorded actions can be saved as XML data files.

In addition, all tools in Sleuth QA Suite 2 feature the ability to export data from KeyViews to XML, HTML, or Microsoft Excel spreadsheet files; support for all 32-bit versions of Borland Delphi, Borland C++Builder, and others; COM Automation support so developers can access and use the debugging and optimization services of the suite from their own programs, without having to use the tool's user interface; TurboPower's Dynamic Instrumentation technology; and an improved user interface with full support for docking and access to syntax-highlighted source code and advanced disassembly listings (where applicable).

TurboPower Software Co.
Price: US$399
Phone: (800) 333-4160
Web Site: http://www.turbopower.com

SkyLine Tools Imaging Releases Doc-to-Net 6.0

SkyLine Tools Imaging announced ***Doc-to-Net 6.0***, a new version of the company's turnkey solution for sharing documents over the Internet by transforming TIFF files to a browser-compliant format.

Doc-to-Net 6.0 is a CGI application that transforms a scanned TIFF image to a PNG, GIF, or JPEG, then anti-aliases it "on the fly" and streams it through the browser. This allows the end user to view the document without downloading any plug-ins. Also included are zoom, pan, and scroll features, plus rotating and inverting controls.

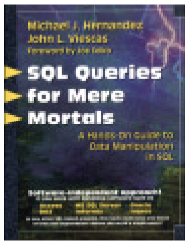
New features in version 6.0 include a COM object for ASP pages, which will return the number of pages in a multi-page TIFF file; the ability to recognize and return sizes of varied size pages within a multi-page TIFF file; fewer required DLLs; a UNC path name feature, allowing files on different computers within a system to be accessed by the same server; and password protection.

For photographic images, Doc-to-Net supports BMP, PCX, and TGA files, and presents them as JPEG, PNG, or GIF files. Image correction tools are provided for images.

Developers who wish to utilize Doc-to-Net as a part of an independent programming application can leverage it as an add-on to SkyLine Tools' Corporate Suite. Doc-to-Net is royalty free.

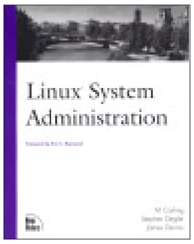
SkyLine Tools Imaging
Price: US$599
Phone: (818) 346-4200
Web Site: http://www.imagelib.com

SQL Queries for Mere Mortals
Michael J. Hernandez and John L. Viescas
Addison-Wesley



ISBN: 0-201-43336-2
Price: US$39.95
Web Site: http://www.ComputerBookstore.com

Linux System Administration
M. Carling, Stephen Degler, and James Dennis
New Riders



ISBN: 1-56205-934-3
Price: US$29.99
Web Site: http://www.ComputerBookstore.com

Quest Announces Optional Module for TOAD

Quest Software, Inc. announced *TOAD DBA*, an optional module for its TOAD application development software that adds database administration functionality to address common Oracle database management tasks.

Utilizing a simple graphical user interface, Oracle database administrators have an easier way to handle database management tasks, such as adding data files, managing disk space, creating or modifying user accounts, implementing database security, and creating or scheduling batch jobs — and they can perform these tasks from within the development environment.

TOAD DBA has a number of specific features that help with the most common database management tasks, including schema compare and synchronize, SQL data load and export, and batch job scheduling and management. It also provides the ability to run DBA reports.

Another significant feature is Tablespace Management. Once users confirm that a particular tablespace is nearly out of disk storage space, they can log into the database, confirm current space allocations, and add disk space as needed.

The original TOAD development software allows developers to build, test, and debug PL/SQL packages, procedures, triggers, and functions, and it allows users to create and edit database tables, views, indexes, constraints, users, and roles. TOAD integrates with version control systems, schema management solutions, and SQL tuning and impact analysis tools.

Quest Software, Inc.
Price: US$495
Phone: (949) 754-8000
Web Site: http://www.quest.com

eHelp Launches RoboHELP Office 9.0

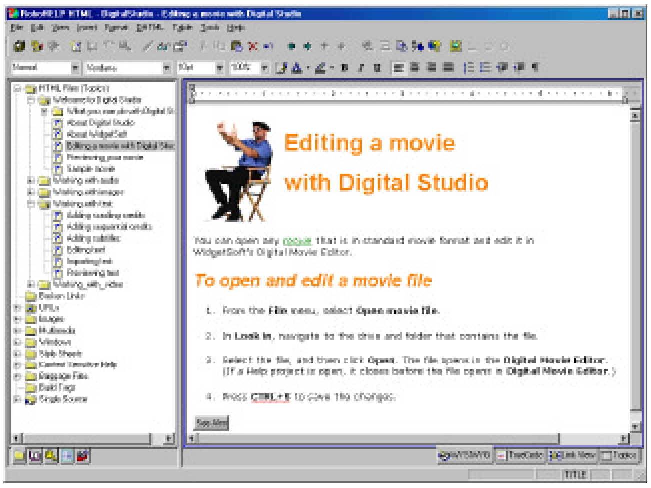
eHelp Corp. announced it is shipping ***RoboHELP Office 9.0***, the latest version of the software tool for developing Help systems. RoboHELP Office 9.0 automates the process of creating Help and assistance for Web sites and Web-based applications, allowing end users to answer questions and resolve problems themselves.

RoboHELP Office uses Help-style navigation and features that users are familiar with from their desktop applications (table of contents, index, full-text search, associative linking, browse sequences, etc.).

RoboHELP supports major online Help formats, including WebHelp, Microsoft HTML Help, WinHelp, JavaHelp, Oracle Help for Java, and more, and allows users to create Help systems for deployment on any platform.

RoboHELP Office 9.0 features include integrated support for popular HTML editors, such as Dreamweaver, FrontPage, and HomeSite; a built-in HTML editor with dozens of new features; and enhanced WebHelp 4.0.

eHelp Corp.
Price: US$899
Phone: (800) 358-9370
Web Site: http://www.ehelp.com/robohelp


Editing a movie with Digital Studio

You can open any movie that is in standard movie format and edit it in WidgetSoft's Digital Movie Editor.

To open and edit a movie file

1. From the **File** menu, select **Open movie file.**
2. In **Look in**, navigate to the drive and folder that contains the file.
3. Select the file, and then click **Open**. The file opens in the **Digital Movie Editor**. (If a Help project is open, it closes before the file opens in **Digital Movie Editor**.)
4. Press CTRL+S to save the changes.

Linux Essential Reference
Ed Petron
New Riders

ISBN: 0-7357-0852-5
Price: US$24.95
Web Site: http://www.ComputerBookstore.com

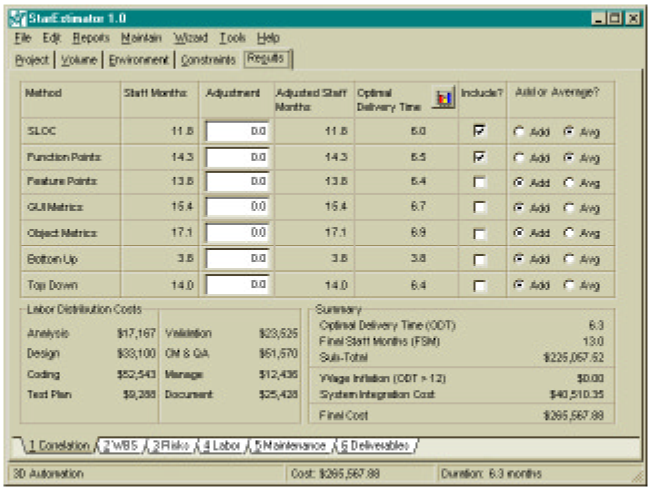
4Tier Software Announces OpenMOM

4Tier Software announced the release of *OpenMOM*, which allows the deployment of component-based distributed Internet applications. With OpenMOM, Delphi, C/C++, Java, and other applications become distributed communicating components deployed within a multi-tier model over the Internet.

OpenMOM manages the exchange and morphing of objects between heterogeneous software components via its asynchronous/synchronous messaging infrastructure, as well as interconnecting message brokers that ensure maximum security, load balancing, and the availability of the distributed information systems.

Some features of OpenMOM include object morphing, dynamic load balancing, multi-threaded architecture, broadcast and multicast messaging, fault tolerant and multiple broker architecture, and GUI-based administration and configuration.

4Tier Software
Price: Free for development and deployment with up to three computers connected.
Phone: (33) 1 39 76 80 20
Web Site: http://www.4tier.com


StaffEstimator 1.0
File Edit Reports Maintain Wizard Tools Help
Project Volume Environment Constraints Results

Method	Staff Months	Adjustment	Adjusted Staff Months	Optimal Delivery Time	Include?	Add or Average?
SLOC	11.8	0.0	11.8	6.0	✓	Add [✓ Avg]
Function Points	14.3	0.0	14.3	6.5	✓	Add [✓ Avg]
Feature Points	13.8	0.0	13.8	6.4		[✓ Add] Avg
GUI Metrics	15.4	0.0	15.4	6.7		[✓ Add] Avg
Object Metrics	17.1	0.0	17.1	6.9		[✓ Add] Avg
Bottom Up	3.8	0.0	3.8	3.8		[✓ Add] Avg
Top Down	14.0	0.0	14.0	6.4		[✓ Add] Avg

Labor Distribution Costs

Analysis	$17,167	Validation	$23,526
Design	$33,100	CM & QA	$51,570
Coding	$52,543	Manage	$12,436
Test Plan	$9,288	Document	$25,428

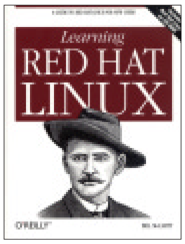
Summary

Optimal Delivery Time (ODT)	6.3
Final Staff Months (FSM)	13.0
Sub-Total	$225,057.52
Wage Inflation (ODT > 12)	$0.00
System Integration Cost	$40,510.35
Final Cost	$265,567.88

1 Correlation 2 WBS 3 Risks 4 Labor 5 Maintenance 6 Deliverables

3D Automation Cost: $265,567.88 Duration: 6.3 months

Learning Red Hat Linux
Bill McCarty
O'Reilly



ISBN: 1-56592-627-7
Price: US$34.95
Web Site: http://www.ComputerBookstore.com

Advanced Software Technologies Unveils GDPro 5.0

Advanced Software Technologies, Inc. announced the release of *GDPro 5.0*, an upgrade to the company's UML visual modeling solution for accelerating software design and collaborative development. This new version allows application developers to create and communicate across the enterprise using simple, comprehensive visual blueprints of Java, C++, and IDL.

New features include a Web System Browser Report, with functionality that allows developers to produce and share comprehensive diagrams of their software designs over the Internet.

Other new features include Java Reverse Engineering performance enhancements, context-sensitive pop-up Windows to speed diagram population, and a Rational Rose diagram import feature.

Advanced Software Technologies, Inc.
Price: US$2,995 for Windows; US$4,795 for Unix.
Phone: (800) 811-2784
Web Site: http://www.advancedsw.com

It's Time to Vote in the *Delphi Informant Magazine* Readers Choice Awards

Elk Grove, CA — Delphi Informant Magazine announced the official ballot for the 2001 Delphi Readers Choice Awards is now available online at http://www.DelphiZine.com.

Each year, *Delphi Informant Magazine* recognizes outstanding products and vendors in the Delphi add-on market that you select. Please take a moment to vote for your favorite Delphi tools in 19 categories, including Best Book, Best Reporting Tool, and Product of the Year. Simply complete the online ballot. This is your chance to voice your opinion regarding the tools and products you use every day.

Voting ends January 31, 2001. The results will be published in the April 2001 issue of *Delphi Informant Magazine*.

Inprise/Borland Announces JBuilder 4

Scotts Valley, CA — Inprise/Borland announced Borland JBuilder 4, the latest version of its pure-Java cross-platform development environment. JBuilder 4 supports Enterprise JavaBean (EJB) 1.1-compliant development on Windows, Linux, and Solaris platforms.

JBuilder 4 streamlines the concurrent management of source code for large distributed teams while extending existing source code management across the Internet.

JBuilder 4 features new visual two-way tools, designers, and wizards, such as the entity Bean modeler, which shortens the development and deployment of e-business applications; InternetBeans, JSP, and servlets for building data-driven Web applications; support for Java 2 version 1.3 with built-in support for the Sun Java HotSpot Virtual Machine; JDataStore 4, a pure-Java Object Relational Database Management System for Web, mobile, and embedded applications; and remote debugging on Windows, Linux, Solaris, or any platform that supports Java 2 JPDA, such as HP-UX, AIX, Tru64.

JBuilder 4 is integrated with the Borland Application Server and supports BEA's WebLogic server. Users can run and debug EJBs and CORBA applications locally or remotely in the JBuilder environment and deploy EJB instantly without shutting down the application server.

JBuilder 4 will be available in three versions: JBuilder 4 Enterprise, JBuilder 4 Professional, and JBuilder 4 Foundation. For more information, visit http://www.borland.com/jbuilder, or call (800) 632-2864.

Inprise/Borland Announces Availability of Inprise Application Server 4.1

Scotts Valley, CA — Inprise/Borland announced the shipment and availability of the Inprise Application Server 4.1. Inprise Application Server provides a foundation for customers to deploy Internet applications with full support for Enterprise Java Bean (EJB) and CORBA technology. This technology allows companies whose business models are based around large volumes of transactions, such as financial institutions, to closely integrate their existing systems with the new EJB environment.

New features of Inprise Application Server 4.1 include Wireless Access Protocol support; enhanced EJB transaction support with full two-phase commit VisiTransact, a transaction manager supporting JDBC 2.0/XA; VisiMessage; advanced object to relational mapping for EJBs; integration support for Thought Inc.'s CocoBase and other third-party O/R mapping tools; an enhanced user interface; and new wizards to simplify rolling EJB applications out to multiple containers.

Inprise Application Server 4.1 is available immediately from local Inprise/Borland offices, distributors, and representatives. For more information on Inprise Application Server, visit http://www.borland.com/appserver.

Inprise/Borland to Support Intel Itanium Processor

San Jose, CA — Inprise/Borland announced it will port Borland JBuilder, Inprise Application Server, VisiBroker for Java, and JDataStore for platforms based on the Intel Itanium processor. The software will be ported to the Microsoft Windows 2000 64-bit and Red Hat Linux 64-bit operating systems running Java 2 v1.3 runtime from IBM. All Inprise/Borland offerings on Intel IA-64 architecture (IA-64) — including Itanium processor-based systems — will be compatible with what's available on IA-32 today. This will enable companies who use Inprise/Borland's integrated Java tool set to easily move their applications running on IA-32-based systems today to IA-64-based systems of tomorrow.

The Intel Itanium processor is the first in a family of IA-64 processors from Intel and is the most significant new development in Intel microprocessor architecture since the 386-processor introduction in 1985. The Itanium processor will complement today's Pentium III processor and Intel Pentium Xeon 32-bit processor families with new features of high-end scalability, reliability, availability, and large memory addressability. More information on the Intel Itanium processor is available at http://developer.intel.com/design/ia-64.

By Binh Ly

Implementing COM+ Events

Exploiting the Windows 2000 COM+ Event System

One of the more interesting features of COM+ (the new version of COM for Microsoft Windows 2000) is the introduction of a new event mechanism for developing scalable distributed systems. To appreciate the COM+ event system, it's important to first understand how events were implemented in the pre-COM+ era.

Pre-COM+ events fall into two main categories:

- COM connection points are a standard way of negotiating event interfaces between the source and the receiver of the events. This structured negotiation is dictated by the architecture of some standard COM interfaces, such as *IConnectionPoint* and *IConnectionPointContainer*.
- COM callbacks are a low-level, application-specific way of negotiating event interfaces between the source and the receiver of the events. This negotiation usually requires a custom architecture, specific to an application or part of an application.

I won't go into the technical details of COM connection points and COM callbacks; that's not the goal of this article. (For more information on these two techniques, refer to my two-part series in the June and July 1998 issues of *Delphi Informant Magazine*.)

One of the major characteristics of pre-COM+ events is that the source and the receiver of events are tightly integrated:

- Before events can be triggered, the event source and receiver must be simultaneously running and available. If either is unavailable, no event can be successfully triggered.
- The event receiver always has "pre-compiled" knowledge of the event source. The event receiver will almost always know which particular COM class to instantiate as the event source, and this knowledge is hard-coded and compiled into the event-receiver application.
- The infrastructure is somewhat biased and caters to a one-to-one correspondence between the event source and receiver. For instance, it can take an effort to implement multiple event receivers (particularly as separate applications) to receive events from a single event source.

Because of these behaviors, the pre-COM+ event infrastructure is often referred to as a tightly coupled event (TCE) system. The term "tightly coupled" denotes the strong and intimate dependency between the event source and receiver objects.

A tightly coupled event system has its place, and has been used successfully over the years, e.g. for implementing the ActiveX controls event mechanism. Unfortunately, with today's applications rapidly moving toward high-speed, enterprise-scale environments, such as Web applications, new event requirements have surfaced that are difficult to satisfy with a TCE solution.

A Hypothetical Retail Application

To introduce some of the requirements of a distributed event system, imagine that we are building the Purchase Order part of a hypothetical retail application named the The Best. (The code for the sample application is available for download; see end of article for details.) Specifically, we're in charge of an Order business component whose interface implements a simple purchase-order placement transaction as follows:

```
IOrder = interface
  procedure PlaceOrder(CustomerID, ProductID,
                       Quantity);
end;
```

CustomerID refers to the customer placing the order, *ProductID* specifies the product being ordered, and *Quantity* specifies how many products are being ordered.

Let's assume our business requirements are as follows:

1) Every time an order transaction is placed, an e-mail message must be sent to a known back-end mail server, where another routine will take control and start the process of filling the order.
2) Every time an order transaction is placed for a particular product (*ProductID*), we'll need to log that into the system. This log is used by other statistical applications that will analyze purchase patterns of customers and experimental products. Experimental products change regularly, so it would be nice to have an implementation for this that isn't unduly impacted by product changes.

Obviously, a brute-force way to implement these requirements is to hard-code the business processes into the *IOrder.PlaceOrder* implementation. However, we can look at these requirements from a different angle, and conclude that we may want to trigger some kind of event every time an order is placed. Other business processes or components can then "listen in" on this event to perform the desired operation, based on our current (and future) business requirements.

Using an event-based approach, it's easy to see that:

- Requirements 1 and 2 don't need to be implemented by a single business application/component; requirement 1 can be implemented by one business component, and requirement 2 by another. The only commonality among the business components is that they both listen in on the same event. Furthermore, future requirements can be implemented by adding newer business components without modifying our existing application.
- Requirements 1 and 2 don't require an application continuously running somewhere "out there," listening for an event; we only need to execute a business process/component whenever an event is triggered.
- One powerful idea would be to allow easy configuration of each business component that listens in on the event, or even the configuration of the event itself. We might need to turn the event on and off at certain times (such as when doing back-end maintenance). We may also need to disable one business process independently of the other. For example, we may need to temporarily disable the business component that handles requirement 2, without affecting the business process handling for requirement 1.

Unfortunately, the nature of pre-COM+ event systems, such as connection points and callbacks, does not meet these requirements. COM+ introduces a new concept, called a loosely coupled event (LCE) system, for such enterprise-scale requirements. It's based on three core concepts: the event class, the event publisher, and the event subscriber.

COM+ Event Infrastructure

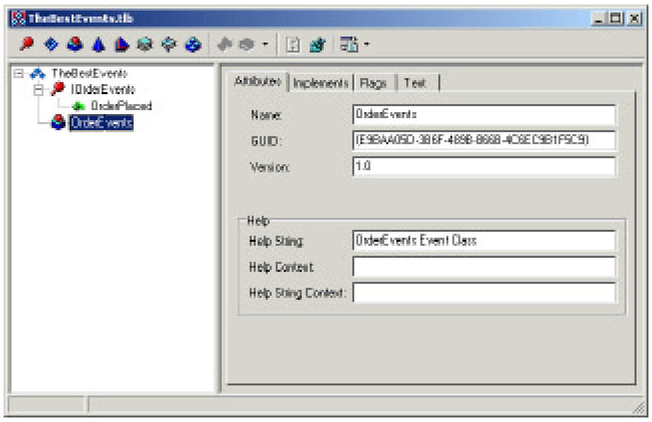
Using our example, we need some sort of concrete entity to represent the event that gets triggered every time *IOrder.PlaceOrder* is executed. This concrete entity is called an *event class*. An event class is nothing more than a simple COM class (a CoClass plus interface) definition of the event. Let's define our *OrderEvents* event class as follows:

```
IOrderEvents = interface
  prodecure OrderPlaced(CustomerID, ProductID, Quantity);
end;

OrderEvents = CoClass
  implements IOrderEvents
end;
```

This representation means that *OrderEvents* is an event class that implements the *IOrderEvents* interface, which contains an *OrderPlaced* event method.

An important thing to understand about event classes is that we never implement the methods of the event class interface. We simply define the interface, and then install it into the COM+ environment (see Figure 1).

To further understand what I mean, let's take a look at the Delphi implementation of an event class. Figure 2 illustrates the *OrderEvents* event class (TheBestEvents.dpr). This module was created using Delphi's Automation Object Wizard (**File | New | ActiveX | Automation Object**) and using "OrderEvents" as the CoClass name.



Name: OrderEvents
GUID: {E9BAA05D-3B6F-489B-B66B-4C8EC9B1F5C9}
Version: 1.0
Help String: OrderEvents Event Class
Help Context:
Help String Context:

Figure 1: The *OrderEvents* event class definition as it appears in the Type Library editor.

```
unit
OrderEvents;

interface

uses
  Windows, ActiveX, Classes, ComObj, TheBestEvents_TLB,
  StdVcl;

// OrderEvents event class.
type
  TOrderEvents = class(TAutoObject, IOrderEvents)
  protected
    procedure OrderPlaced(
      const CustomerID, ProductID: WideString;
      Quantity: Integer); safecall;
  end;

implementation

uses
  ComServ;

procedure TOrderEvents.OrderPlaced(const CustomerID,
  ProductID: WideString; Quantity: Integer);
begin
  // Event class methods are not implemented.
end;

initialization
  TAutoObjectFactory.Create(ComServer, TOrderEvents,
    Class_OrderEvents, ciMultiInstance, tmApartment);
end.
```

Figure 2: This module describes the *OrderEvents* event class.

At this point, you might be wondering how this event is physically triggered. The answer is simple: We create an instance of the *OrderEvents* CoClass, and call the *OrderPlaced* method from that instance:

```
uses
  TheBestEvents_TLB;

procedure TriggerEvent;
var
  OrderEvents: IOrderEvents;
begin
  // Create OrderEvents event class.
  OrderEvents := CoOrderEvents.Create;
  // Execute OrderEvents.OrderPlaced event.
  OrderEvents.OrderPlaced('A Customer', 'A Product', 5);
end;
```

To put that in context, let's trigger this event from within the *PlaceOrder* method of the Order business component, as shown in Figure 3.

The process of triggering a COM+ event works only if we properly install the *OrderEvents* event class into the COM+ environment. The process of installing an event class for everybody to see is known as *publishing* the event. The party that publishes an event is called the *event publisher*. In our case, the publisher of the *OrderEvents* event class is our retail application, The Best.

Publishing Events

An event class can be published administratively (interactively) or programmatically. I will only demonstrate administrative publishing; programmatic publishing requires knowledge of programmatic access to the COM+ administration APIs that is beyond the scope of this article. Before we can publish an event class, we first need to build the COM server that defines the event class. This is contained in the project, TheBestEvents.dpr. Building this project produces TheBestEvents.dll, which we'll install into the COM+ environment.

First, we'll need to run the Component Services administration tool, and create a new COM+ application in it. Component Services is accessible from the Windows 2000 Control Panel, under Administrative Tools | Component Services. To do this, right-click on COM+ Applications and select New | Application from the pop-up menu. The COM Application Install Wizard will be displayed.

Select Create an empty application, and click on Next to advance to the Create Empty Application screen. Then enter "The Best" as the name for the new application, select Server application as the Activation type, and click on Next to advance to the Set Application Identity screen. Select Interactive user as the Account and click on Next. Click Finish on the next screen to complete the wizard and return to Component Services, where The Best will now appear as a new, empty COM+ application (see Figure 4).

The next step is to install *OrderEvents* as an event class into application, The Best. To do this, right-click on Components under The Best, and select New | Component. The COM Component Install Wizard will be displayed (see Figure 5). Select Install new event class(es) and click Next. Browse to the TheBestEvents.dll file, and select it to advance to the Install new event class screen. Click Next, then Finish to complete the wizard.

We've learned what an event class is, how it's installed, and how it's called to trigger events. The next step is to understand how we can listen in on an event.

Subscribing to Events

Listening for a COM+ event is known as *subscribing* to the event. The party that subscribes to a COM+ event is called an *event subscriber*. An event subscriber is physically implemented as a class that implements

```
uses
  TheBestEvents_TLB;

procedure TOrder.PlaceOrder(
  const CustomerID, ProductID: WideString;
  Quantity: Integer);
var
  OrderEvents: IOrderEvents;
begin
  // Step 1:
  // Perform PlaceOrder business logic here.
  // Example: Update Order table in database.

  // Step 2:
  // Trigger OrderEvents.
  // Create OrderEvents event class.
  OrderEvents := CoOrderEvents.Create;
  // Execute OrderEvents.OrderPlaced event.
  OrderEvents.OrderPlaced(CustomerID, ProductID, Quantity);
end;
```

Figure 3: Triggering the event from the *PlaceOrder* method of the *TOrder* class.

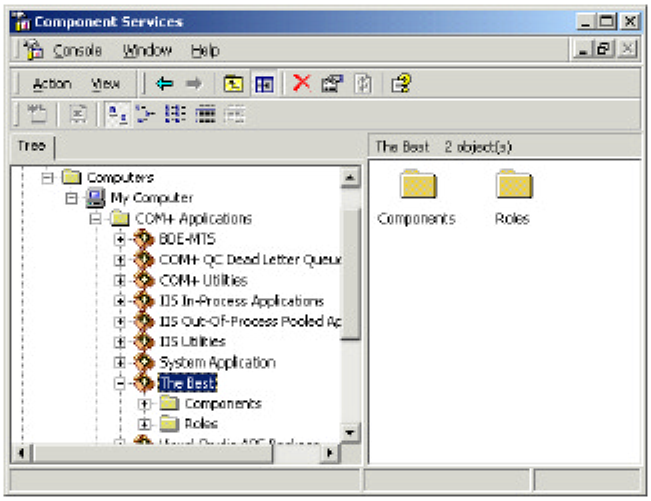


Figure 4: Component Services after creating an empty COM+ application, The Best.

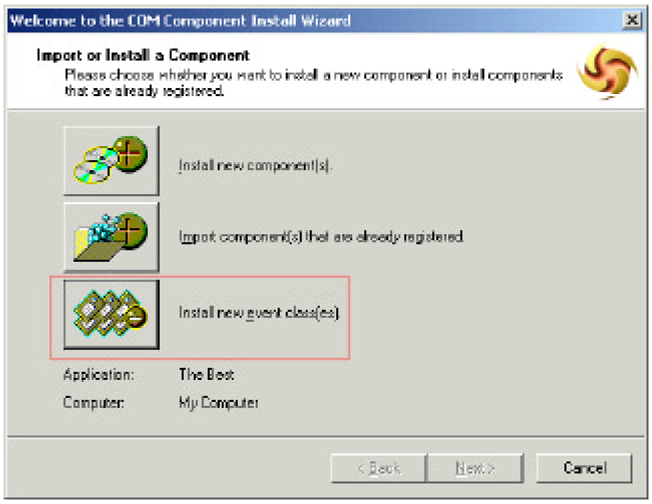

Welcome to the COM Component Install Wizard
Import or Install a Component
Please choose whether you want to install a new component or install components that are already registered.
Install new component(s).
Import component(s) that are already registered.
Install new event class(es).
Application: The Best
Computer: My Computer
< Back
Next >
Cancel

Figure 5: The COM Component Install Wizard.

the event interface(s) of a COM+ event class. A subscriber component can be a persistent or a transient subscriber. A *persistent subscriber* is one that can be installed and administered via the COM+ environment. It doesn't need to be running to receive event notifications. In contrast, a *transient subscriber* cannot be administered via the COM+ environment, and must be running to receive event notifications.

For our purposes, we'll build two persistent subscribers that implement the two requirements mentioned earlier in this article. The first subscriber implements sending out an e-mail notification every time *IOrderEvents.OrderPlaced* is triggered. Let's call this subscriber *EMailer*. *EMailer* is implemented in TheBestSubscribers.dpr, as shown in Figure 6. This module was created using the Delphi Automation Object Wizard and using "EMailer" as the CoClass name.

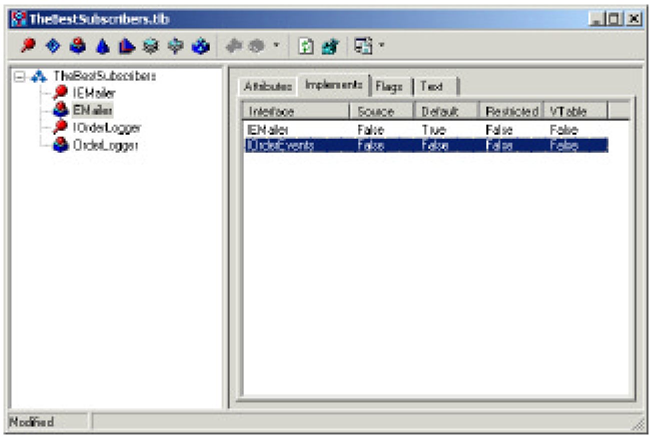
It's important to point out that *EMailer* implements the *IOrderEvents* event interface. More importantly, this needs to be reflected in the type library information for *EMailer*. To do this, view the TheBestSubscribers.tlb in the Delphi Type Library editor, and select the Uses page. Right-click to display the pop-up menu, and select **Show All Type Libraries**. Page down to **TheBest Event Classes** and select it.

Now we need to associate the *IOrderEvents* interface with the *EMailer* CoClass. To do this, click on the **EMailer** CoClass (while still in the Type Library editor), and select the Implements page. Right-click to display the pop-up menu, and select **Insert Interface**. Select **IOrderEvents** from the displayed dialog box and click **OK**. The result is shown in Figure 7.

```
unit EMailer;

interface

uses
  ComObj, ActiveX, TheBestSubscribers_TLB,
  TheBestEvents_TLB, StdVcl;

// EMailer subscriber.
type
  TEMailer = class(TAutoObject, IEMailer, IOrderEvents)
  protected
    procedure OrderPlaced(
      const CustomerID, ProductID: WideString;
      Quantity: Integer); safecall;
    procedure SendEMail;
  end;

implementation

uses
  ComServ, Dialogs, SysUtils;

// IOrderEvents.OrderPlaced subscription implementation.
procedure TEMailer.OrderPlaced(const CustomerID,
  ProductID: WideString; Quantity: Integer);
begin
  SendEMail;
  ShowMessage('EMail Processing completed!' + #13 +
    'Customer: ' + CustomerID + ', Product: ' + ProductID +
    ', Quantity: ' + IntToStr(Quantity));
end;

procedure TEMailer.SendEMail;
begin
  // Do e-mail business logic here.
end;

initialization
  TAutoObjectFactory.Create(ComServer, TEMailer,
    Class_EMailer, ciMultiInstance, tmApartment);
end.
```

Figure 6: *EMailer* subscriber code.

Building the *OrderLogger* subscriber is very similar to EMailer. *OrderLogger* is also implemented in TheBestSubscribers.dpr, as shown in Figure 8. This module was created using the Automation Object



Interface	Source	Default	Restricted	VTable
EMailer	False	True	False	False
IOrderEvents	False	False	False	False

Figure 7: The *EMailer* CoClass in the Type Library editor.

```
unit OrderLogger;

interface

uses
  ComObj, ActiveX, TheBestSubscribers_TLB,
  TheBestEvents_TLB, StdVcl;

// Order Log subscriber.
type
  TOrderLogger = class(TAutoObject, IOrderLogger,
                       IOrderEvents)
  protected
    procedure OrderPlaced(
      const CustomerID, ProductID: WideString;
      Quantity: Integer); safecall;
    procedure LogOrder;
  end;

implementation

uses
  ComServ, Dialogs, SysUtils;

// IOrderEvents.OrderPlaced subscription implementation.
procedure TOrderLogger.OrderPlaced(
  const CustomerID, ProductID: WideString;
  Quantity: Integer);
begin
  LogOrder;
  ShowMessage('Order logging completed!' + #13 +
    'Customer: ' + CustomerID + ', Product: ' + ProductID +
    ', Quantity: ' + IntToStr(Quantity));
end;

procedure TOrderLogger.LogOrder;
begin
  // Do logging business logic here.
end;

initialization
  TAutoObjectFactory.Create(ComServer, TOrderLogger,
    Class_OrderLogger, ciMultiInstance, tmApartment);
end.
```

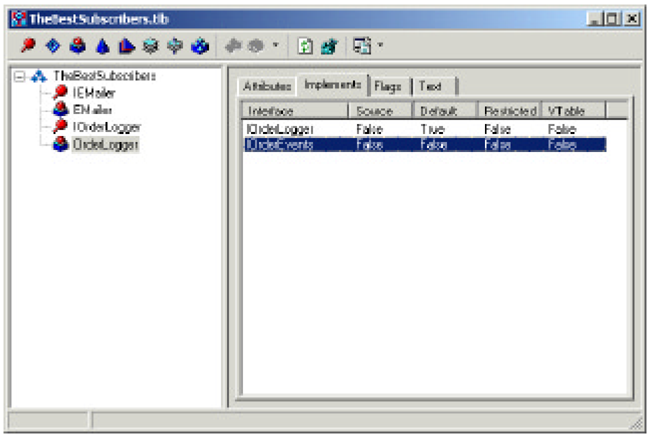
Figure 8: The *OrderLogger* unit.

wizard and using "OrderLogger" as the CoClass name. Also, note the important type library information included with the *OrderLogger* CoClass, as shown in Figure 9.

With these, we are now ready to install *EMailer* and *OrderLogger* into the COM+ environment. To do this, we first build TheBestSubscribers.dpr, creating TheBestSubscribers.dll. We then install both components into a COM+ application using the Component Services utility. For our purposes, let's install these into the application we created earlier, i.e. TheBest.

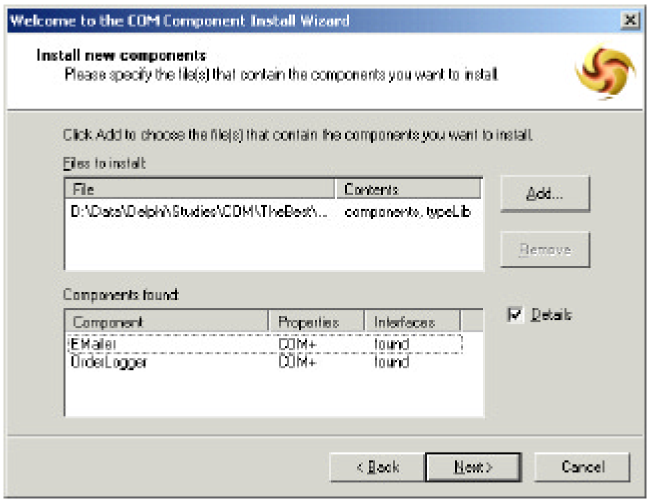
To do this, right-click on **Components** under **The Best** in Components Services, and select **New | Component**. This will invoke the COM Component Install Wizard. Select **Install new component(s)** at the Import or Install a Component screen, and click **Next**. Browse to TheBestSubscribers.dll file and select it to advance to the Install new components screen (see Figure 10).

After installation, we then configure the event subscriptions. To configure *EMailer*'s subscriptions, right-click on **Subscriptions** under **TheBestSubscribers.EMailer** and select **New | Subscription** (as shown in Figure 11) to display the COM New Subscription Wizard. The Select Subscription Method(s) screen will be displayed; select **IOrderEvents** and click **Next**. Then select **TheBestEvents.OrderEvents** at the select Event Class screen and click **Next**. Make sure the **Enable this subscription immediately** option is checked on the Subscription Options screen, click **Next**, then **Finish** on the following screen to complete the wizard.



Interface	Source	Default	Restricted	VTable
IOrderLogger	False	True	False	False
IOrderEvents	False	False	False	False

Figure 9: The *OrderLogger* definition in the Type Library editor.


Welcome to the COM Component Install Wizard

Install new components
Please specify the file(s) that contain the components you want to install.

Click Add to choose the file(s) that contain the components you want to install.

Files to install:

File	Contents
D:\Data\Delphi\Studies\COM\TheBest\...	components, typeLib

Components found:

Component	Properties	Interfaces
EMailer	COM+	found
OrderLogger	COM+	found

☑ Details

Figure 10: Installing *EMailer* and *OrderLogger* subscribers.

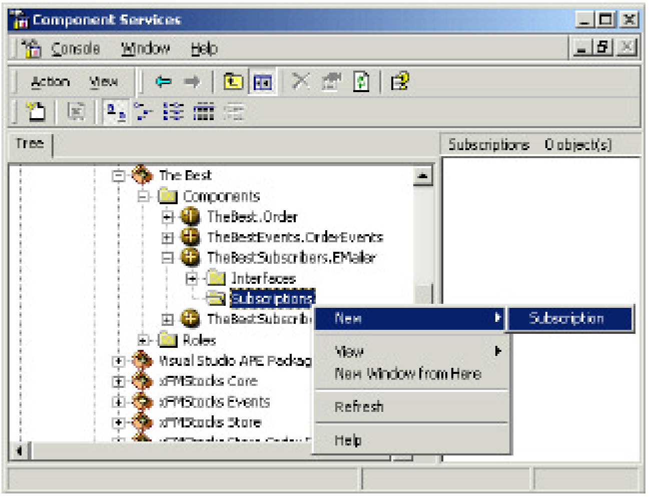


Figure 11: Invoking the COM New Subscription Wizard.

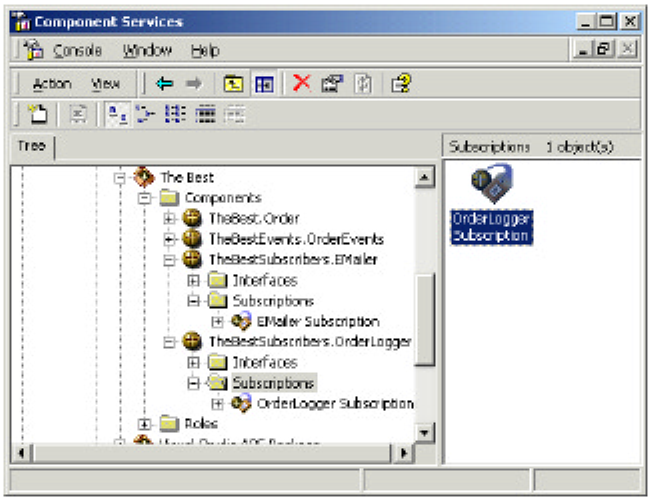
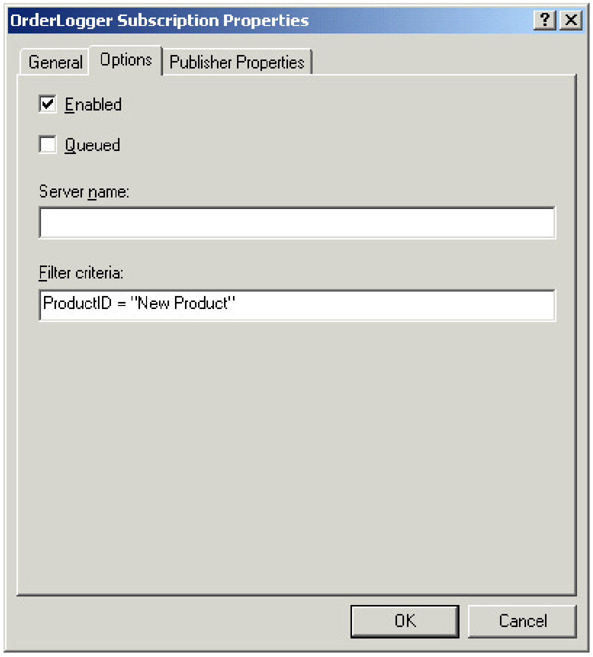


Figure 12: Component Services after configuring the subscriptions for *EMailer* and *OrderLogger*.


OrderLogger Subscription Properties

General | Options | Publisher Properties

☑ Enabled

☐ Queued

Server name:

Filter criteria: ProductID = "New Product"

Figure 13: Configuring *OrderLogger's* subscription filter.

Next, perform the same process for *OrderLogger* subscription. When you're done, Component Services should look like Figure 12.

Subscriber Filters

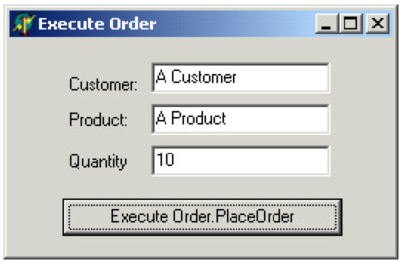
Recall that one requirement for *OrderLogger* was that we want it to log transactions only for certain products. Although this business requirement can be realized by implementing conditional business logic in *TOrderLogger.OrderPlaced*, we want the ability to configure this product filter administratively, so we can easily change the filtered product without a large maintenance impact on *OrderLogger*. Fortunately, the COM+ event system has just the facility we need: subscriber filters. A *subscriber filter* is a simple Boolean expression that's evaluated to determine if a subscriber should be notified of certain events. As an example, let's assume we want to log orders for an experimental product named "New Product." This is easily configured, as shown in Figure 13.

Note that Filter criteria uses the *ProductID* taken from the specified parameter name in the *IOrderEvents.OrderPlaced* method. The filter criteria consists of a string Boolean expression. The expression is evaluated by the COM+ event system before instantiation of the subscriber component. This provides for an efficient filtering mechanism in which subscribers aren't unnecessarily instantiated if they aren't "interested" in an event, based on certain conditions. The filter criteria can also include Boolean operators, such as OR, AND, and NOT.

A Sample Application

Now that we've installed both subscriber business components, we can then test their functionality. Before we do this, however, let's install the Order business component (discussed earlier) that triggers the events. To do this, build the TheBest.dpr project and install TheBest.dll into the COM+ environment. On my machine, I simply added this into The Best.

To test the events, I've built a simple client application (ExecuteOrder.dpr) that simulates a purchase order transaction (see Figure 14). Performing a purchase order transaction is implemented as shown in Figure 15.



Customer:	A Customer
Product:	A Product
Quantity	10

Figure 14: A simple purchase-order client application.

Assuming you've installed the event classes and subscriptions correctly, executing the *Order.PlaceOrder* operation should produce a message box from the *EMailer* subscriber. And changing the Product edit box contents to "New Product" should produce a message box from the *OrderLogger* subscriber.

```
uses
  TheBest_TLB;

procedure TForm1.ExecuteOrderClick(Sender: TObject);
var
  Order: IOrder;
begin
  // Create Order business component.
  Order := CoOrder.Create;
  // Execute Order.PlaceOrder business transaction.
  Order.PlaceOrder(CustomerID.Text, ProductID.Text,
    StrToInt(Quantity.Text));
end;
```

Figure 15: Performing a purchase order transaction.

Conclusion

This article has walked through a simple example for which the COM+ event system is a perfect solution. We've only scratched the surface of COM+ events, however. I hope this serves as a good introduction, and piques your interest enough for you to delve more deeply into the details of implementing COM+ events in your own applications. Δ

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\DEC\DI200012BL.

Binh Ly is an independent consultant who specializes in developing distributed systems using COM technologies. He also maintains a Web site (http://www.techvanguards.com) that educates people on how to effectively use COM technologies using Borland products. You can reach Binh at bly@techvanguards.com.

By Jeremy Merrill

New List Objects

Delphi 5's New Classes Increase *TList* Class Abilities

Delphi 5 introduced a new Contnrs unit that defines eight new classes, all based on the standard *TList* class. In this article, we'll provide an overview of the *TList* class, then explore these eight new classes in detail. We'll also look at how to create custom list classes, ending with the creation of a new *TList* descendant class, *TIntList*.

The *TList* Class

For those unfamiliar with a *TList* object, it simply stores a list of pointers. In many ways it's like a dynamic array, with properties and methods that allow addition, deletion, rearrangement, location, access, and sorting of items within the list. *TList* is often used to maintain a list of objects (see the tables in Figures 1, 2, and 3).

The *TObjectList* Class

The primary class definition in the Contnrs unit is *TObjectList*, which descends directly from the *TList* class (see Figure 4). Given that most *TList* objects are used to store a list of objects, this new class may be the most useful contribution of the Contnrs unit.

The first thing to notice about *TObjectList* is that *Add*, *Remove*, *IndexOf*, *Insert*, and *Items* all expect a type of *TObject* instead of Pointer. This allows for more strict type checking at compile time, and removes the need to type cast item pointers as *TObjects*.

The second thing to notice is the addition of the *OwnsObjects* Boolean property. This property is the fundamental difference between *TList* and *TObjectList*. When True (the default value), this property directs *TObjectList* to free any objects removed from its list. This includes the use of the *Delete*, *Remove,* and *Clear* methods, as well as when the *TObjectList* is itself destroyed. It's even smart enough to free an object that has been replaced by the insertion of a new object into an existing list position.

Name	Type	Description
Count: Integer;	property	Returns the number of items in the list.
Items[Index: Integer]: Pointer; default	property	Allows access to items in a zero-based list.

Figure 1: Commonly used public properties of the *TList* class.

Anyone who's used *TList* to maintain a list of objects will appreciate this feature. No more loops to free objects before freeing the *TList* object. No more freeing individual objects before their removal. No more memory leaks caused by bugs in the freeing code. This feature is so important to this class that there's even an overloaded *Create* constructor that allows the setting of the *OwnsObjects* property when the *TObjectList* object is created.

One final note on this topic: When *OwnsObjects* is True, the *Extract* method will remove the object reference from the list, but will not free the object. The *Extract* method was added to *TList* in Delphi 5 for this purpose, even though the *TList* implementation has *Extract* and *Remove* performing essentially the same function.

The last thing to notice about *TObjectList* is the *FindInstanceOf* method. This function returns the index of an object in the list that has the same class as the one passed as the argument for *AClass*. If *AExact* is True, only instances of that class are found. If *AExact* is False, objects that inherit from *AClass* will also be found. The *AStartAt* parameter can be used to find multiple instances of a class within the list, continually calling the *FindInstanceOf* method with a starting index value one greater than the index of the object last found, until *FindInstanceOf* returns a -1. This is illustrated in the following code:

```
var
  idx: Integer;
begin
  idx := -1;
  repeat
    idx := ObjList.FindInstanceOf(TMyObject,
             True, idx+1);
    if idx >= 0 then
      ...
  until(idx < 0);
end;
```

The *TComponentList* Class

The second class definition in the Contnrs unit is *TComponentList* (see Figure 5). Notice that *TComponentList* descends from *TObjectList*, giving it all the capabilities of that class. Also notice that *Add*, *Remove*, *IndexOf*, *Insert,* and *Items* have all been changed to use a *TComponent* type. *TComponentList* has one additional feature that is not obvious from looking at the public class definition. If a component in the list is freed (perhaps by its owning form), then it will automatically be removed from the *TComponentList* list. This is accomplished by using the *TComponent.FreeNotification* method, which causes components in the list to notify the *TComponentList* object that they are about to be destroyed.

Name	Type	Description
Add(Item: Pointer): Integer;	function	Used to add a new item to the end of the list.
Clear;	procedure	Used to clear out the list and reset the *Count* to zero.
Delete(Index: Integer);	procedure	Used to remove a specific item by index.
IndexOf(Item: Pointer): Integer;	function	Used to find the list index of a specific item.
Insert(Index: Integer; Item: Pointer);	procedure	Used to insert an item into the list at a specific index.
Remove(Item: Pointer): Integer;	function	Used to remove a specific item from the list, without needing to know the item's index.

Figure 2: Commonly used public methods of the *TList* class.

Name	Type	Description
Capacity: Integer;	property	Allows precise control over memory allocation for list.
Extract(Item: Pointer): Pointer;	function	*Extract* is identical to *Remove* except it returns a reference to the item; may take on different behavior.
Exchange(Index1, Index2: Integer);	procedure	Used to swap two items in the list.
First: Pointer;	function	Returns the first item in the list.
Last: Pointer;	function	Returns the last item in the list.
Move(CurIndex NewIndex: Integer);	procedure	Used to reposition item from indexes.
Pack;	procedure	Removes all **nil** pointer elements from the list.
Sort(Compare: TListSortCompare);	procedure	Used to sort items in a list. Passed *Compare* routine is used to determine item order.

Figure 3: Some of the less common properties and methods of the *TList* class.

```
TObjectList = class(TList)
  ...
public
  constructor Create; overload;
  constructor Create(AOwnsObjects: Boolean); overload;
  function Add(AObject: TObject): Integer;
  function Remove(AObject: TObject): Integer;
  function IndexOf(AObject: TObject): Integer;
  function FindInstanceOf(AClass: TClass;
    AExact: Boolean = True; AStartAt: Integer = 0):
    Integer;
  procedure Insert(Index: Integer; AObject: TObject);
  property OwnsObjects: Boolean;
  property Items[Index: Integer]: TObject; default;
end;
```

Figure 4: Class declaration for *TObjectList*.

```
TComponentList = class(TObjectList)
  ...
public
  function Add(AComponent: TComponent): Integer;
  function Remove(AComponent: TComponent): Integer;
  function IndexOf(AComponent: TComponent): Integer;
  procedure Insert(Index: Integer; AComponent: TComponent);
  property Items[Index: Integer]: TComponent; default;
end;
```

Figure 5: Class declaration for *TComponentList*.

The *TClassList* Class

The third class definition in Contnrs is *TClassList*, shown in Listing One (beginning on page 15). Unlike the previous two classes, this descendant of *TList* doesn't add functionality. It simply changes *Add*, *Remove*, *IndexOf*, *Insert*, and *Items* to use type *TClass*. While this list would prove useful in certain programming situations, it's perhaps just as useful as an example of how to create your own custom *TList* descendant class. I'm including the entire class definition here to illustrate how the supporting methods simply call the inherited implementation of the same method.

To create a *TMyObjectList* class, which descended from either *TList* or *TObjectList*, you would simply change all the references of *TClass* to *TMyObject*, as shown in Listing Two (on page 16). Of course, once you have the basic framework of a *TMyObjectList* class in place, adding methods or properties that use specific functionality of *TMyObject* would be a natural extension of the class, as demonstrated by the *DoSomething* procedure shown in Listing Two.

TOrderedList, TStack, and *TQueue* Classes

Now we'll look at three class definitions in the Contnrs unit at the same time, *TOrderedList*, *TStack,* and *TQueue*, as shown in Figure 6. Notice that although *TOrderedList* does not descend from *TList*, it uses a *TList* object internally to store a list of items. Also notice that because the protected *PushItem* procedure is declared as abstract, the *TOrderedList* class is an abstract class.

```
TOrderedList = class(TObject)
private
  FList: TList;
protected
  procedure PushItem(AItem: Pointer); virtual; abstract;
  ...
public
  function Count: Integer;
  function AtLeast(ACount: Integer): Boolean;
  procedure Push(AItem: Pointer);
  function Pop: Pointer;
  function Peek: Pointer;
end;

TStack = class(TOrderedList)
protected
  procedure PushItem(AItem: Pointer); override;
end;

TQueue = class(TOrderedList)
protected
  procedure PushItem(AItem: Pointer); override;
end;
```

Figure 6: *TOrderedList*, *TStack*, and *TQueue*.

Although Delphi prohibits creating an instance of an abstract class, *TStack* and *TQueue* are two descendant classes you can create instances of. These two classes differ only in their implementation of the protected *PushItem* procedure. As their class names suggest, *TStack* stores items in a last-in-first-out manner (or LIFO), and *TQueue* stores items in a first-in-first-out manner (or FIFO). Here's a brief look at the individual methods of both classes:

- *Count* returns the number of items in the list.
- *AtLeast* can be used to check the size of the list; it returns True if the number of items in the list is greater than or equal to the passed value.
- For *TStack*, *Push* adds an item to the end of the list. For *TQueue*, *Push* inserts an item at the beginning of the list.
- *Pop* returns an item from the end of the list, and removes it from the list.
- *Peek* returns an item from the end of the list, but leaves it in the list.

TObjectStack and *TObjectQueue* Classes

Now we'll look at the last two class definitions in the Contnrs unit, *TObjectStack* and *TObjectQueue*, shown in Listing Three (on page 16). These two classes are simple extensions of the *TStack* and *TQueue* classes, containing a list of *TObjects* instead of pointers. Unlike the *TObjectList* class presented earlier, these classes do not offer any new functionality. In most situations it wouldn't make any sense for these lists to own their objects, because popping an object off the list would result in its destruction. These classes use the same techniques as shown in the *TClassList* definition, calling inherited methods while converting between objects and pointers. A close examination of these two classes demonstrates how easy it would be to create a *TMyObjectStack* or *TMyObjectQueue* class, as shown in Listing Four (beginning on page 16).

The *TIntList* Class

Up to this point, all the list classes we have looked at have been based on a list of pointers. The Object and Component lists are really lists of pointers, which just happen to point to *TObjects* or *TComponents*. We've even discussed how to create custom *TMyObject* classes, which would essentially be a list of pointers that just happen to point to *TMyObjects*.

Now we're going to define our own *TList* class that, instead of containing a list of pointers, contains a list of integers. We'll do this by creating a new IntList unit, which defines the *TIntList* class, shown in Listing Five (on page 17). This class makes use of the fact that pointers and integers both take the same amount of storage space — four bytes. By typecasting pointers as integers, we can store an integer value in the space reserved for a pointer. All the supporting methods for this class simply call the inherited method or property, and include one or more typecast conversions between pointers and integers. (The IntList unit is available for download; see end of article for details.)

When looking at the *TIntList* class, notice that, in addition to the *Add*, *Remove*, *IndexOf*, *Insert,* and *Items* redefinitions, we also redefine *First*, *Last,* and *Extract*. The reason for this is that in *TObjectList*, *TComponentList*, and *TClassList*, these methods would still function, because these classes are still dealing with pointers. In *TIntList*, these methods would no longer function unless they were redefined as using integers. Also notice the *Sort* method, which makes use of the *TList* sorting capabilities by passing the *IntListCompare* function to the inherited *Sort* method.

Conclusion

Delphi 5 has provided us with several new classes that extend the capabilities of the *TList* class. The *TObjectList* and *TComponentList* classes allow us to store lists of objects or components without having to write additional supporting code. The *TStack* and *TQueue* classes provide a more structured approach to using lists. While *TClassList*, *TObjectStack,* and *TObjectQueue* are only simple extensions of these classes, we've examined how to use these examples to create custom list classes based on our own *TMyObject* class. We've even defined a new *TIntList* class that extends the capabilities of the *TList* class to contain a list of integers. While the main intent of this article has been to showcase the new classes provided in the Contnrs unit, I also hope that I have instilled a better understanding of how you can create custom list classes. ∆

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\DEC\DI200012JM.

Jeremy Merrill is an EDS contractor in a partnership contract with the Veteran's Health Administration. He is a member of the VA's Computerized Patient Record System development team, located in the Salt Lake City Chief Information Officer's Field Office.

Begin Listing One — *TClassList*

```
TClassList = class(TList)
protected
  function GetItems(Index: Integer): TClass;
  procedure SetItems(Index: Integer; AClass: TClass);
public
  function Add(aClass: TClass): Integer;
  function Remove(aClass: TClass): Integer;
  function IndexOf(aClass: TClass): Integer;
  procedure Insert(Index: Integer; aClass: TClass);
  property Items[Index: Integer]: TClass
    read GetItems write SetItems; default;
end;
...
{ TClassList }
function TClassList.Add(AClass: TClass): Integer;
begin
  Result := inherited Add(AClass);
end;

function TClassList.GetItems(Index: Integer): TClass;
begin
  Result := TClass(inherited Items[Index]);
end;

function TClassList.IndexOf(AClass: TClass): Integer;
begin
  Result := inherited IndexOf(AClass);
end;

procedure TClassList.Insert(Index: Integer;
  AClass: TClass);
begin
  inherited Insert(Index, AClass);
end;

function TClassList.Remove(AClass: TClass): Integer;
```

```
begin
  Result := inherited Remove(AClass);
end;

procedure TClassList.SetItems(Index: Integer;
  AClass: TClass);
begin
  inherited Items[Index] := AClass;
end;
```

End Listing One

Begin Listing Two — *TMyObjectList*

```
TMyObject = class(TObject)
public
  procedure DoSomething;
end;

TMyObjectList = class(TObjectList)
protected
  function GetItems(Index: Integer): TMyObject;
  procedure SetItems(Index: Integer; AMyObject: TMyObject);
public
  function Add(aMyObject: TMyObject): Integer;
  procedure DoSomething;
  function Remove(aMyObject: TMyObject): Integer;
  function IndexOf(aMyObject: TMyObject): Integer;
  procedure Insert(Index: Integer; aMyObject: TMyObject);
  property Items[Index: Integer]: TMyObject
    read GetItems write SetItems; default;
end;
...
{ TMyObjectList }
function TMyObjectList.Add(AMyObject: TMyObject): Integer;
begin
  Result := inherited Add(AMyObject);
end;

procedure TMyObjectList.DoSomething;
var
  i: Integer;
begin
  for i := 0 to Count-1 do
    Items[i].DoSomething;
end;

function TMyObjectList.GetItems(Index: Integer): TMyObject;
begin
  Result := TMyObject(inherited Items[Index]);
end;

function TMyObjectList.IndexOf(AMyObject: TMyObject):
  Integer;
begin
  Result := inherited IndexOf(AMyObject);
end;

procedure TMyObjectList.Insert(Index: Integer;
  AMyObject: TMyObject);
begin
  inherited Insert(Index, AMyObject);
end;

function TMyObjectList.Remove(AMyObject: TMyObject):
  Integer;
begin
  Result := inherited Remove(AMyObject);
end;

procedure TMyObjectList.SetItems(Index: Integer;
  AMyObject: TMyObject);
begin
  inherited Items[Index] := AMyObject;
end;
```

End Listing Two

Begin Listing Three — *TObjectStack* and *TObjectQueue*

```
TObjectStack = class(TStack)
public
  procedure Push(AObject: TObject);
  function Pop: TObject;
  function Peek: TObject;
end;

TObjectQueue = class(TQueue)
public
  procedure Push(AObject: TObject);
  function Pop: TObject;
  function Peek: TObject;
end;
...
{ TObjectStack }
function TObjectStack.Peek: TObject;
begin
  Result := TObject(inherited Peek);
end;

function TObjectStack.Pop: TObject;
begin
  Result := TObject(inherited Pop);
end;

procedure TObjectStack.Push(AObject: TObject);
begin
  inherited Push(AObject);
end;

{ TObjectQueue }
function TObjectQueue.Peek: TObject;
begin
  Result := TObject(inherited Peek);
end;

function TObjectQueue.Pop: TObject;
begin
  Result := TObject(inherited Pop);
end;

procedure TObjectQueue.Push(AObject: TObject);
begin
  inherited Push(AObject);
end;
```

End Listing Three

Begin Listing Four — *TMyObjectStack* and *TMyObjectQueue*

```
TMyObjectStack = class(TStack)
public
  procedure Push(AMyObject: TMyObject);
  function Pop: TMyObject;
  function Peek: TMyObject;
end;

TMyObjectQueue = class(TQueue)
public
  procedure Push(AMyObject: TMyObject);
  function Pop: TMyObject;
  function Peek: TMyObject;
end;
...
```

```
{ TMyObjectStack }
function TMyObjectStack.Peek: TMyObject;
begin
  Result := TMyObject(inherited Peek);
end;

function TMyObjectStack.Pop: TMyObject;
begin
  Result := TMyObject(inherited Pop);
end;

procedure TMyObjectStack.Push(AMyObject: TMyObject);
begin
  inherited Push(AMyObject);
end;

{ TMyObjectQueue }
function TMyObjectQueue.Peek: TMyObject;
begin
  Result := TMyObject(inherited Peek);
end;

function TMyObjectQueue.Pop: TMyObject;
begin
  Result := TMyObject(inherited Pop);
end;

procedure TMyObjectQueue.Push(AMyObject: TMyObject);
begin
  inherited Push(AMyObject);
end;
```

End Listing Four

Begin Listing Five — IntList.pas

```
unit IntList;

interface

uses
  Classes;

type
  TIntList = class(TList)
  protected
    function GetItem(Index: Integer): Integer;
    procedure SetItem(Index: Integer;
      const Value: Integer);
  public
    function Add(Item: Integer): Integer;
    function Extract(Item: Integer): Integer;
    function First: Integer;
    function IndexOf(Item: Integer): Integer;
    procedure Insert(Index, Item: Integer);
    function Last: Integer;
    function Remove(Item: Integer): Integer;
    procedure Sort;
    property Items[Index: Integer]: Integer
      read GetItem write SetItem; default;
  end;

implementation

{ TIntList }
function TIntList.Add(Item: Integer): Integer;
begin
  Result := inherited Add(Pointer(Item));
end;

function TIntList.Extract(Item: Integer): Integer;
begin
  Result := Integer(inherited Extract(Pointer(Item)));
end;

function TIntList.First: Integer;
begin
  Result := Integer(inherited First);
end;

function TIntList.GetItem(Index: Integer): Integer;
begin
  Result := Integer(inherited Items[Index]);
end;

function TIntList.IndexOf(Item: Integer): Integer;
begin
  Result := inherited IndexOf(Pointer(Item));
end;

procedure TIntList.Insert(Index, Item: Integer);
begin
  inherited Insert(Index, Pointer(Item));
end;

function TIntList.Last: Integer;
begin
  Result := Integer(inherited Last);
end;

function TIntList.Remove(Item: Integer): Integer;
begin
  Result := inherited Remove(Pointer(Item));
end;

procedure TIntList.SetItem(Index: Integer;
  const Value: Integer);
begin
  inherited Items[Index] := Pointer(Value);
end;

function IntListCompare(Item1, Item2: Pointer): Integer;
begin
  if Integer(Item1) < Integer(Item2) then
    Result := -1
  else if Integer(Item1) > Integer(Item2) then
    Result := 1
  else
    Result := 0;
end;

procedure TIntList.Sort;
begin
  inherited Sort(IntListCompare);
end;

end.
```

End Listing Five

By Alex Fedorov

Two Office Web Components

Working with the OWC Chart and Spreadsheet Controls

One of the new and exciting features of Microsoft Office 2000 is its Office Web Components (OWC), a set of ActiveX components intended for use from HTML pages to create "active" Web documents. Since these components are simply ActiveX components, however, we can also use them in non-Web applications built with Delphi. This article shows you how.

The six ActiveX components that comprise the OWC implement some of the functionality of Microsoft Excel, and provide database access. The components are named ChartSpace, Spreadsheet, DataSourceControl, RecordNavigationControl, ExpandControl, and PivotTable. DataSourceControl and RecordNavigationControl can be replaced with ADOExpress VCL components (as we'll see in this article), so they're of no particular interest to Delphi developers. ExpandControl is also outside the scope of this article.

This leaves ChartSpace, Spreadsheet, and PivotTable. Unfortunately, I haven't found a direct way to make the PivotTable component work in Delphi 5; it cannot be activated correctly for some reason. We can use still the component in Delphi, however, and in a future issue of this magazine we'll see the PivotTable component at work, and discuss accessing the Microsoft Excel Pivot Table services.

As with many programming services available through a type library — ActiveX components, Automation servers, etc. — we need to create a wrapper unit to use them. (This step is unnecessary if you're using late binding. In that case, you can use Windows COM API functions to create objects and variants to use the methods and properties implemented in them. However, the easier way to deal with ActiveX components is to create a wrapper unit.) To do so, select **Project | Import Type Library** from the Delphi menu, and select **Microsoft Office Web Components 9.0 (Version 1.0)** from the list of available type libraries.

Since we're dealing with ActiveX components, we need to check the **Generate Component Wrapper** option, select the **Palette page** where Delphi will put the components wrappers (the default is ActiveX), and click **Install**. The six components will be added to the Component palette. As already mentioned, we'll discuss only the ChartSpace and Spreadsheet components.

The type library file for the OWC is named MSOWC.DLL, and lives by default in \Program Files\Microsoft Office 2000\Office. The help file for the OWC is named MSOWCVBA.CHM, and lives in \Program Files\Microsoft Office 2000\Office\1033 by default.

The ChartSpace Component

As its name denotes, ChartSpace is used to create charts. ChartSpace supports 46 types of charts, from Line to Doughnut Exploded. See the *ChartChartTypeEnum* constants in OWC help for the complete list. It can also draw charts from other members of the OWC (the Spreadsheet component in our case), from ADO data sources, or from static data. In this article, we'll demonstrate how to do all three. (The demonstration projects are available for download; see end of article for details.)

Charts Based on Static Data

Static data is supplied by the application, i.e. it's

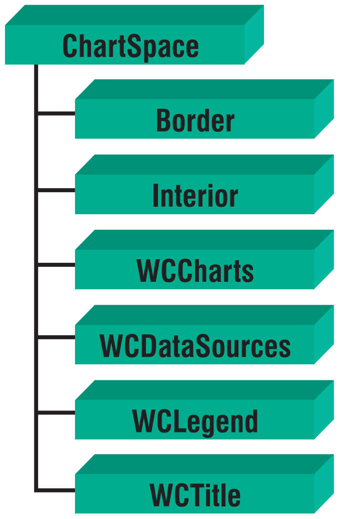


Figure 1: Simplified ChartSpace object model.

hard wired. In Delphi applications, we store such data in static variant arrays created with the *VarArrayCreate* function, or in dynamic variant arrays created with the *VarArrayOf* function.

Variant arrays. Let's take a brief refresher course on the use of variant arrays. Static variant arrays hold items of type Variant. Such arrays are created with the *VarArrayCreate* function, and are filled on an item-by-item basis, or in a loop. The following example shows how to create and fill a static variant array:

```
var
  XValues : Variant;
  ...
  // Create the variant array.
  XValues := VarArrayCreate([0,2], varVariant);
  // Load with data for the X-axis.
  XValues[0] := 'Element one';
  XValues[1] := 'Element two';
  XValues[2] := 'Element three';
```

When we call the *VarArrayCreate* function, we indicate the dimensions of the variant array and the type of its items, *varVariant* in this case. In the example above, we've created a variant array with three items of type Variant.

A dynamic variant array is created with the *VarArrayOf* function, and isn't associated with a particular variable. In the following example, the *VarArrayOf* function returns a one-dimensional array of items of type Variant:

```
VarArrayOf([104737, 50952, 78128, 117797,
            52902, 80160, 47491, 62435]));
```

Now, let's return to the ChartSpace component.

To use the ChartSpace component we must perform several steps. First, we have to clear the contents of the chart. Next, we need to add a new Chart object to the *Charts* collection, which can hold up to 16 charts. After that we can supply data for series (in the *Series* collection), set the title (the *Title* property), specify the axes (in the *Axes* collection), and legend (the *Legend* property). Figure 1 shows a simplified object hierarchy of the ChartSpace component (see OWC Help for the complete diagram).

Instead of giving you a detailed description of the ChartSpace component, and its objects and collections, let's just look at an example that uses static data to create a chart. First, we need some variables:

```
var
  Chart   : WCChart;   // Chart
  Series  : WCSeries;  // Series
  XValues : Variant;   // X-axis values
```

Now to manipulate the component. As indicated above, we need to clear the contents of the chart before we can use it:

```
ChartSpace1.Clear;
ChartSpace1.Refresh;
```

Let's create a new chart. To do this we need to add an item to the *Charts* collection. Since this is the first chart, it has an index value of 0.

```
Chart := ChartSpace1.Charts.Add(0);
```

Now, let's add the title for the chart. First, we need to indicate that our chart will have a title, then supply the text for the title.

```
Chart.HasTitle := True;
Chart.Title.Caption := 'Sales By Category';
```

After that, we need to supply data. But first, because we're using the static data in this example, let's create it. For the X-axis, we will use a static variant array:

```
XValues := VarArrayCreate([0,7], varVariant);
XValues[0] := 'Beverages';
XValues[1] := 'Condiments';
XValues[2] := 'Confections';
XValues[3] := 'Dairy Products';
XValues[4] := 'Grains & Cereals';
XValues[5] := 'Meat & Poultry';
XValues[6] := 'Produce';
XValues[7] := 'Seafood';
```

Now we can use this data to build our chart. The first step is to add the new *Series* object to the *SeriesCollection* of our Chart (see Figure 2).

To illustrate, let's add another series, and put the new chart on top of the existing one, as shown in Figure 3.

```
Series := Chart.SeriesCollection.Add(0);
with Series do begin
  // Set Series caption...
  Caption := '1998';
  // ...and data.
  SetData(chDimCategories, chDataLiteral, XValues);
  // Y-axes data will be taken from dynamic variant array.
  SetData(chDimValues, chDataLiteral, VarArrayOf(
    [104737, 50952, 78128, 117797,
     52902,  80160, 47491, 62435]));
  // Set Chart type, Clustered Column in this case.
  Type_ := chChartTypeColumnClustered;
end;
```

Figure 2: Building a chart.

```
Series := Chart.SeriesCollection.Add(1);
with Series do begin
  // Set Series caption...
  Caption := '1999';
  // ...and data.
  SetData(chDimCategories, chDataLiteral, XValues);
  // Y-axes data is taken from the dynamic variant array.
  SetData(chDimValues, chDataLiteral, VarArrayOf(
          [20000, 15000, 36000, 56000,
           40000, 18000, 20000, 33000]));
  // Set the Chart type, e.g. Line Markers.
  Type_ := chChartTypeLineMarkers;
end;
```

Figure 3: Adding another chart to *SeriesCollection*.

To complete our first charting example, we just need to add an axis:

```
Chart.Axes.Add(Chart.Axes[chAxisPositionLeft].Scaling,
  chAxisPositionRight, chValueAxis);
```

and set some of its properties:

```
Chart.Axes[chAxisPositionLeft ].NumberFormat := '$#,##0';
Chart.Axes[chAxisPositionRight].NumberFormat := '0';
Chart.Axes[chAxisPositionLeft ].MajorUnit    := 20000;
Chart.Axes[chAxisPositionRight].MajorUnit    := 20000;
```

Then we add a legend at the bottom part of our chart:

```
Chart.HasLegend := True;
Chart.Legend.Position := chLegendPositionBottom;
```

The resulting chart is shown in Figure 4. So, we've seen how to use static data to draw charts with the ChartSpace component. In the next example, we will use data stored in a database.

Charts Based on Data from a Database

When we need to build a chart based on data stored in a database, we have two options. We can use DataSource from the OWC, or we can use some ADO-compatible data source. As we will see later in this article, the main sequence of actions is the same; the only difference is how the data sources are defined.

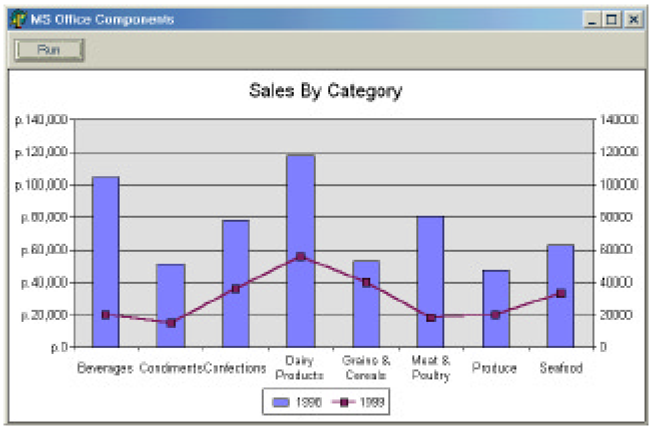


Figure 4: This line chart was created with static (hard wired) data.

```
// Set the horizontal layout for the chart.
ChartSpace1.ChartLayout := chChartLayoutHorizontal;
// Add new Chart.
BarChart := ChartSpace1.Charts.Add(0);
with BarChart do begin
  // Set the type of Chart, a bar chart.
  Type_ := chChartTypeBarClustered;
  // The first field is categories.
  SetData(chDimCategories, 0, 0);
  // The second field is values.
  SetData(chDimValues, 0, 1);
  // Format the axes.
  with Axes[chAxisPositionBottom] do begin
    NumberFormat      := '0,';
    MajorUnit         := 25000;
    HasMajorGridlines := False;
  end;
end;
```

Figure 5: Creating a bar chart.

Using DataSource from the OWC

Let's start with the OWC DataSource component. First, we'll declare some variables:

```
var
  RSD      : RecordsetDef;  // Datasource
  BarChart : WCChart;       // Chart
  PieChart : WCChart;       // Chart
```

Next, we'll define the data source and the SQL statement used to extract the data. For this example, we're using the sample Northwind database that comes with Microsoft Access (Northwind.mdb). Specifically, we're selecting data from the Category Sales for 1997 view:

```
begin
  DataSourceControl1.ConnectionString :=
    'DRIVER={Microsoft Access Driver (*.mdb)};' +
    'DBQ=C:\DATA\NORTHWIND.MDB';
  RSD := DataSourceControl1.RecordsetDefs.AddNew(
    'SELECT * FROM [Category Sales for 1997]', 3, 'Sales');
```

Then, we clear the chart, and indicate the source of data:

```
with ChartSpace1 do begin
  Clear;
  Refresh;
  DataSource := DataSourceControl1.DefaultInterface as
                  MSDATASRC_TLB.DataSource;
  DataMember := RSD.Name;
end;
```

Because the *DataSource* property of the ChartSpace component is declared as an *IUnknown*-based interface (see the MSDATASRC_TLB unit), we need to use the construction shown here. (In Visual Basic, the statement is simpler — we just assign the *DataSourceControl1* value to the *DataSource* property. In this case, the default interface will be found automatically.)

Using an ADO Data Source

Now, let's see how we can use an ADO-compatible data source. To do this, we'll use Delphi's ADOExpress components, i.e. components on the ADO page of the Component palette. The ADOConnection component is used to set the data source and cursor type, and the ADOCommand component is used to extract data with a SQL query. First, we declare a variable for the ADO recordset:

```
var
  RS : _Recordset;  // ADO RecordSet
```

Then, the following code executes the SQL query, and associates the returned data with the Chart:

```
RS := ADOCommand1.Execute;
with ChartSpace1 do begin
  Clear;
  Refresh;
  DataSource := RS as MSDATASRC_TLB.DataSource;
  DataMember := '';
end;
```

Creating the Chart

After the source of the data is specified — with the OWC Data-Source, or through an ADO recordset — we can create our chart. To do so, we must add a new chart to the *Charts* collection, specify the kind of data we will use, and add the axes. The code that does the job is shown in Figure 5.

Now, let's add another chart — a pie chart — based on the same data, as shown in Figure 6. The resulting chart is shown in Figure 7.

Charts Based on Spreadsheet Data

In our third example, we'll create a chart based on data stored in an OWC Spreadsheet component. When these two OWC ActiveX controls are used together like this, the chart will change its appearance whenever the data in the spreadsheet is changed.

First, we need some variables:

```
var
  Sheet : WorkSheet;  // Spreadsheet
  Chart : WCChart;    // Chart
```

Now we can fill the spreadsheet with some data:

```
Sheet := Spreadsheet1.ActiveSheet;
with Sheet do begin
  Range['A1', 'A10'].Set_Formula('=Row()');
  Range['B1', 'B10'].Set_Formula('=A1^2');
  Range['A12','A12'].Set_Formula('=Max(A1:A10)');
  Range['B12','B12'].Set_Formula('=Max(B1:B10)');
end;
```

In the code above, we have inserted row numbers (from 1 to 10) in the column A and its squared values in the appropriate column B. The cells A12 and B12 contain the maximum values of the columns A and B.

As you can see, the Spreadsheet component allows us to use formulas, which many users will recognize from using Microsoft Excel. You can treat the Spreadsheet component like "Excel Lite;" they share the same binary kernel. Knowing this allows us to use the Spreadsheet component to perform some background calculations, without showing the component itself on the screen.

```
// Add new chart.
PieChart := ChartSpace1.Charts.Add(1);
with PieChart do begin
  // Set the type of chart, a pie chart.
  Type_ := chChartTypePie;
  // The first field is categories.
  SetData(chDimCategories, 0, 0);
  // The second field is values.
  SetData(chDimValues, 0, 1);
  // "Explode" segments of the pie.
  SeriesCollection.Item[0].Explosion := 20;
  // Add the legend...
  HasLegend := True;
  Legend.Position := chLegendPositionBottom;
  // ...and title.
  HasTitle := True;
  Title.Caption := 'Sales by Category for 1997';
  Title.Font.Set_Bold(True);
  Title.Font.Set_Size(11);
  WidthRatio := 75;
  // Data will be shown as percents.
  with SeriesCollection.Item[0].DataLabelsCollection.Add do
    begin
      HasValue := False;
      HasPercentage := True;
      Font.Set_Size(7);
    end
...
```

Figure 6: Adding a pie chart to the collection.

After we've inserted data into the cells, we can use it to draw a chart. First, let's clear the current chart and add a new Chart, then associate it with data — this time with data stored in a Spreadsheet component (see Figure 8).

Note that as in the previous examples in this article, we use the *MSDATASRC_TLB.DataSource* type to specify the default interface for the Spreadsheet component. Now we can "beautify" our chart, by specifying titles for the axes and its types, as shown in Figure 9.

After that, we can specify the maximum and minimum values for axes, and set additional styles:

```
Scalings[chDimXValues].Maximum :=
  Sheet.Range['A12', 'A12'].Value;
Scalings[chDimXValues].Minimum := 1;
Scalings[chDimYValues].Maximum :=
  Sheet.Range['B12', 'B12'].Value;
// Set additional styles.
with SeriesCollection.Item[0] do begin
  Marker.Style := chMarkerStyleDot;
  Marker.Size  := 6;
  Line.Set_Weight(1);
end
```

The resulting chart is shown in Figure 10.

Again, this chart is "live;" if we change data in the Spreadsheet component, it will be automatically reflected in the ChartSpace component.

Now that we've seen how to use a Spreadsheet component as a source of data for OWC charts, in the remaining part of this article, let's see how to use the Spreadsheet component by itself.

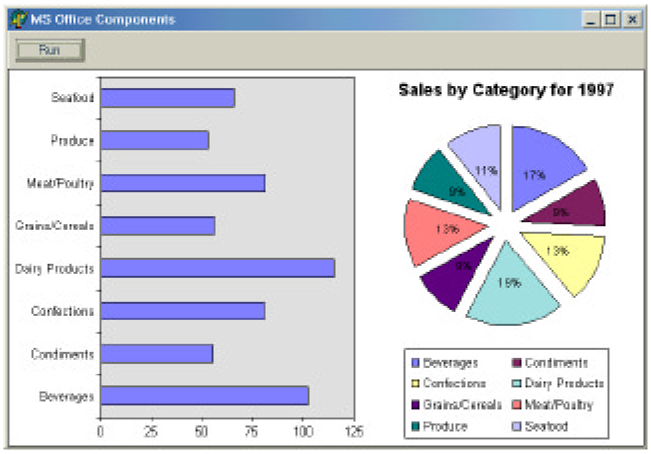


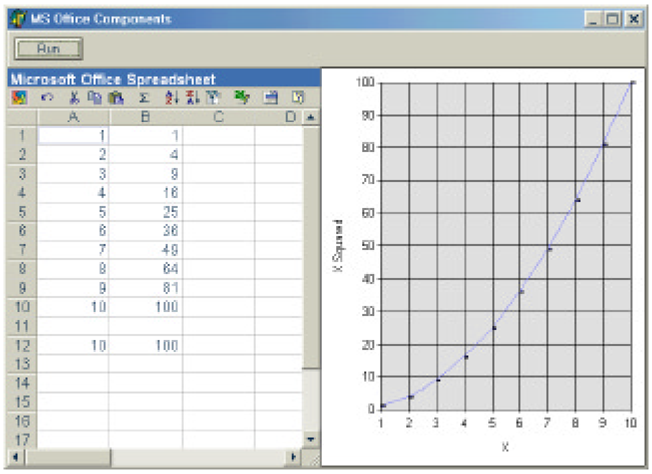
Figure 7: The two charts created in Figures 5 and 6.

```
with ChartSpace1 do begin
  Clear;
  Refresh;
  // Specify the datasource.
  DataSource := Sheet.Parent as MSDATASRC_TLB.DataSource;
  // Add new Chart.
  Chart := Charts.Add(0);
  // Set the type of Chart, Scattered Smooth Line Markers.
  Chart.Type_ := chChartTypeScatterSmoothLineMarkers;
  // Specify data for X...
  Chart.SetData(chDimXValues, 0, 'A1:A10');
  // ...and Y values as a range of cells.
  Chart.SetData(chDimYValues, 0, 'B1:B10');
end;
```

Figure 8: Using a Spreadsheet component as the data source.

```
// Shows titles for the axes.
with Chart do begin
  with Axes[chAxisPositionBottom] do begin
    HasTitle := True;
    Title.Caption := 'X';
    Title.Font.Set_Size(8);
    MajorUnit := 1;
  end;
  with Axes[chAxisPositionLeft] do begin
    HasTitle := True;
    Title.Caption := 'X Squared';
    Title.Font.Set_Size(8);
    MajorUnit := 10;
  end;
...
```

Figure 9: Specifying titles for the chart's axes.



	A	B	C	D
1	1	1		
2	2	4		
3	3	9		
4	4	16		
5	5	25		
6	6	36		
7	7	49		
8	8	64		
9	9	81		
10	10	100		
11				
12	10	100		
13				
14				
15				
16				
17				

Figure 10: A Spreadsheet component, and a chart using its data.

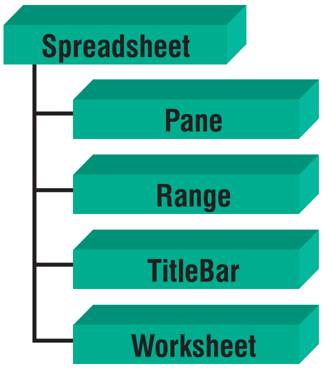


Figure 11: Simplified object model for the Spreadsheet component.

The Spreadsheet Component

The Spreadsheet component provides us with a programmable kernel, which can be used to perform various calculations with a huge library of functions. It also comes with an easy-to-use graphical interface for manipulating the cells of the spreadsheet.

The object model of the Spreadsheet component (see Figure 11) contains the *ActiveSheet* property, which gives us access to an individual sheet within the component; and such objects as *Pane* (the work area of the spreadsheet), *Range* (for a range of cells), *TitleBar* (the title bar for the spreadsheet), and *Worksheet* (for the spreadsheet itself). Again, see OWC Help for the complete diagram.

There are many ways to insert data into the Spreadsheet component. For example, we can enter it manually, use copy and paste (the Clipboard), load data from Microsoft Excel or Word, load data stored in text file or Web site, etc.

To load data from a text file, we can use the Spreadsheet's *LoadText* method, which takes the following arguments:

- the name and path of the file as a string;
- the symbol to use as the delimiter, e.g. tab, comma, etc.;
- a Boolean value that indicates how to treat consecutive delimiters; and
- the text qualifier, which is a double quote by default.

This *LoadText* statement, for example, will load data from a file named Employee.txt, using the tab character (ASCII 9) as the delimiter:

```
SpreadSheet1.LoadText(
  'c:\data\Employee.txt', Chr(9), False, '"');
```

After calling this method, our spreadsheet will be filled with the data contained in the specified text file. Alternatively, we can specify data through the *CSVData* property (for comma-separated data), or through the *HTMLData* property (for HTML-based data). Note that HTML-based data is stored in Microsoft Excel-compatible format. This should be kept in mind if you need to exchange data between the Spreadsheet component and Excel.

In its current version, the Spreadsheet component doesn't feature database support per se. However, it's easy to load data from an ADO-compatible data source, as shown in Figure 12.

In this example, we've used Delphi's ADOCommand component to extract data from the Employees table in the Northwind database. After that, we iterated through the records, and extracted data into the cells of the Spreadsheet component.

After specifying the data, we can set properties for the cells. This example changes the font of the first row:

```
with Spreadsheet1.Range[Spreadsheet1.Cells.Item[1, 1],
     Spreadsheet1.Cells.Item[1, RS.Fields.Count]].Font do
  begin
    Set_Name('Arial Narrow');
    Set_Bold(True);
    Set_Size(11);
  end;
```

Here, we specify that the cells automatically adjust their size to their contents, and to left-align the contents of the cells:

```
with Spreadsheet1.Range[Spreadsheet1.Cells.Item[1, 1],
     Spreadsheet1.Cells.Item[NumRecs, RS.Fields.Count]] do
  begin
    AutoFitColumns;
    Set_HAlignment(ssHAlignLeft);
  end;
```

The resulting spreadsheet is shown in Figure 13.

Setting Colors

Before we end our discussion of Microsoft Office Web Components, let's briefly talk about setting colors. If you try to change the color of an element of a Chart or Spreadsheet component, you'll find that the *Color* property is read-only. This is due to a bug in the Delphi type-library parser.

To solve this problem, we must use the more round-about *Set_Color* method. Another nuisance is that this method takes an argument of type *POleVariant1*. A closer inspection of the type-library interface code shows that this type is defined as a pointer to OLEVariant. The following code shows how to use the *Set_Color* method:

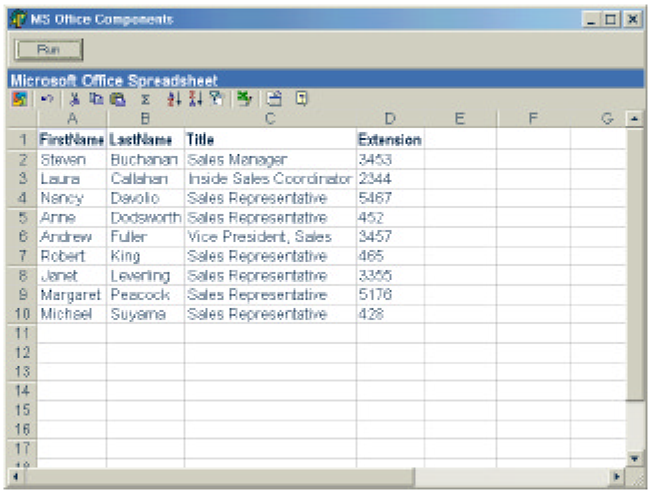
```
var
  C : OLEVariant;
...
// We can use predefined colors...
C := OLEVariant('CornSilk');
// ...or 16-bit RGB values.
C := OLEVariant(RGB($C0, $C0, $C0));
...
Spreadsheet1.ActiveSheet.UsedRange.Interior.Set_Color(@C);
```

Conclusion

So there you have it. We've seen how to use two Microsoft Office Web Components in Delphi applications, so you can add two more ActiveX components to your bag of tricks.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  RS      : _RecordSet;  // ADO Recordset
  I,J     : Integer;     // Counters
  NumRecs : Integer;     // Number Of Records
begin
  // Execute SQL query.
  RS := ADOCommand1.Execute;
  // Move to the first record.
  RS.MoveFirst;
  // Clear the spreadsheet.
  Spreadsheet1.ActiveSheet.Cells.Item[1, 1].Select;
  Spreadsheet1.ActiveSheet.UsedRange.Clear;
  // Set the titles for columns.
  J := 0;
  for I := 0 to RS.Fields.Count-1 do begin
    Inc(J);
    Spreadsheet1.ActiveSheet.Cells.Item[1, J].Set_Value(
      RS.Fields[I].Name)
  end;
  // Set the data.
  I := 1;
  while NOT RS.EOF do begin
    for J := 1 to RS.Fields.Count do
      Spreadsheet1.ActiveSheet.Cells.Item[I+1, J].
        Set_Value(VarToStr(RS.Fields[J-1].Value));
    // Move to the next record.
    RS.MoveNext;
    Inc(I)
  end;
  NumRecs := I;
end;
```

Figure 12: Loading data from an ADO-compatible data source into a Spreadsheet component.



	A	B	C	D	E	F	G
1	FirstName	LastName	Title	Extension			
2	Steven	Buchanan	Sales Manager	3453			
3	Laura	Callahan	Inside Sales Coordinator	2344			
4	Nancy	Davolio	Sales Representative	5467			
5	Anne	Dodsworth	Sales Representative	452			
6	Andrew	Fuller	Vice President, Sales	3457			
7	Robert	King	Sales Representative	465			
8	Janet	Leverling	Sales Representative	3355			
9	Margaret	Peacock	Sales Representative	5176			
10	Michael	Suyama	Sales Representative	428			
11							
12							
13							
14							
15							
16							
17							

Figure 13: A sample Spreadsheet component.

It's important to note that these components can only be used on computers with one of the following products installed: Office 2000 Standard, Office 2000 Premium, Office 2000 Professional, Office 2000 Small Business, Office 2000 Developer, or Access 2000. According to the Microsoft Licensing Agreement, we can use these components only on the local computers and Intranet. For more information refer to http://www.microsoft.com/Office/evaluation/prodinfo/license.htm. Δ

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\DEC\DI200012AF.

Alex Fedorov is executive editor for *ComputerPress* magazine published in Moscow, Russia. He is co-author of *Professional Active Server Pages 2.0* and *ASP 2 Programmer's Reference* published by Wrox, as well as *Advanced Delphi Developer's Guide to ADO*, published by Wordware Publishing. You can visit his Web site at http://d5ado.homepage.com.

By Bruno Sonnino

Fancy Menus, etc.

Custom Menus, Rotated Text, and Special Lines

Before Delphi 4, it was difficult to customize a menu (add a bitmap, change a font, etc.), because owner drawing (i.e. custom drawing) — although implemented by Windows — was not exposed by the *TMainMenu* class. Since Delphi 4, however, this situation has been rectified, and we can have our way with menus.

This article will highlight some techniques you can use to customize the appearance of menus in your Delphi applications. We'll discuss text placement, menu sizing, font assignment, and using bitmaps and shapes to enhance a menu's appearance.

Just for fun, this article also features techniques for creating rotated text and custom lines. All of the techniques discussed in this article are demonstrated in projects available for download; see end of article for details.

Custom Fonts and Sizes

To create a custom menu, set the *OwnerDraw* property of the menu component — *TMainMenu* or *TPopupMenu* — to True, and provide event handlers for its *OnDrawItem* and *OnMeasureItem* events. For example, an *OnMeasureItem* event handler is declared like this:

```
procedure TForm1.Option1MeasureItem(
  Sender: TObject; ACanvas: TCanvas;
  var Width, Height: Integer);
```

Set the *Width* and *Height* variables to adjust the size of the menu item. The *OnDrawItem* event handler is where all the hard work is done; it's where you draw your menu and make any special settings. To draw the menu option with Times New Roman font, for example, you should do something like this:

```
procedure TForm1.Times1DrawItem(
  Sender: TObject; ACanvas: TCanvas;
  ARect: TRect; Selected: Boolean);
begin
  ACanvas.Font.Name := 'Times New Roman';
  ACanvas.TextOut(ARect.Left+1, ARect.Top+1,
    (Sender as TMenuItem).Caption);
end;
```

This code is flawed, however. If it's run, the menu caption will be drawn aligned with the left border of the menu. This isn't default Windows behavior; usually, there's a space to put bitmaps and checkmarks in the menu. Therefore, you should calculate the space needed for a

```
procedure TForm1.Times2DrawItem(Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
var
  dwCheck : Integer;
  MenuCaption : string;
begin
  // Get the checkmark dimensions.
  dwCheck := GetSystemMetrics(SM_CXMENUCHECK);
  // Adjust left position.
  ARect.Left := ARect.Left + LoWord(dwCheck) + 1;
  MenuCaption := (Sender as TMenuItem).Caption;
  // The font name is the menu caption.
  ACanvas.Font.Name := 'Times New Roman';
  // Draw the text.
  DrawText(ACanvas.Handle, PChar(MenuCaption),
           Length(MenuCaption), ARect, 0);
end;
```

Figure 1: This *OnDrawItem* event handler places menu item text correctly.

checkmark with code like that shown in Figure 1. Figure 2 shows the resulting menu.

If the text is too large to be drawn in the menu, Windows will cut it to fit. Therefore, you should set the menu item size so all the text can be drawn. This is the role of the *OnMeasureItem* event handler shown in Figure 3.

Custom Shapes and Bitmaps

It's also possible to customize menu items by including bitmaps or other shapes. To add a bitmap, simply assign a bitmap file to the *TMenuItem.Bitmap* property — with the Object Inspector at design time, or with code at run time. To draw colored rectangles as the caption of a menu item, you could use the *OnDrawItem* event handler shown in Figure 4. Figure 5 shows the result.

There's just one catch. If you're using Delphi 5, you must set the menu's *AutoHotkeys* property to *maManual*. If you leave it as the default, *maAutomatic*, Delphi will add an ampersand character (&) to the caption, which will break this code. Another solution is to remove the ampersand with the *StripHotKey* function.

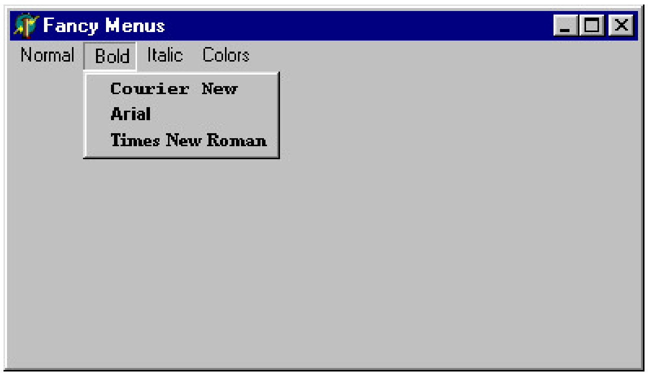


Figure 2: A menu drawn with custom fonts.

```
procedure TForm1.Times2MeasureItem(Sender: TObject;
  ACanvas: TCanvas; var Width, Height: Integer);
begin
  ACanvas.Font.Name := 'Times New Roman';
  ACanvas.Font.Style := [];
  // The width is the space of the menu check
  // plus the width of the item text.
  Width := GetSystemMetrics(SM_CXMENUCHECK) +
    ACanvas.TextWidth((Sender as TMenuItem).Caption) + 2;
  Height := ACanvas.TextHeight(
    (Sender as TMenuItem).Caption) + 2;
end;
```

Figure 3: This *OnMeasureItem* event handler insures that an item fits in its menu.

```
procedure TForm1.ColorDrawItem(Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
var
  dwCheck : Integer;
  MenuColor : TColor;
begin
  // Get the checkmark dimensions.
  dwCheck := GetSystemMetrics(SM_CXMENUCHECK);
  ARect.Left := ARect.Left + LoWord(dwCheck);
  // Convert the caption of the menu item to a color.
  MenuColor :=
    StringToColor((Sender as TMenuItem).Caption);
  // Change the canvas brush color.
  ACanvas.Brush.Color := MenuColor;
  // Draws the rectangle. If the item is selected,
  // draw a border.
  if Selected then
    ACanvas.Pen.Style := psSolid
  else
    ACanvas.Pen.Style := psClear;
  ACanvas.Rectangle(ARect.Left, ARect.Top,
                    ARect.Right, ARect.Bottom);
end;
```

Figure 4: Using the *OnDrawItem* event to draw colored rectangles on menu items.

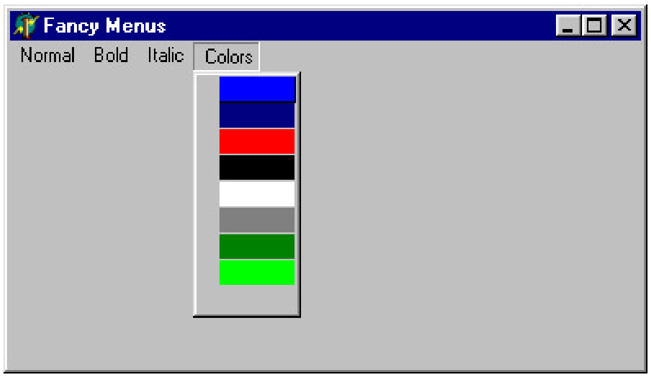


Figure 5: A menu featuring colored rectangles as items.

```
procedure TForm1.VerticalDrawItem(Sender: TObject;
  ACanvas: TCanvas; ARect: TRect; Selected: Boolean);
var
  lf : TLogFont;
  OldFont : HFont;
  clFore, clBack : LongInt;
  Rectang : TRect;
  dwCheck : LongInt;
  MenuHeight : Integer;
begin
  dwCheck := GetSystemMetrics(SM_CXMENUCHECK);
  // This will be done once, when the item is selected.
  if Selected then begin
    // Create a rotated font.
    FillChar(lf, SizeOf(lf), 0);
    lf.lfHeight := -14;
    lf.lfEscapement := 900;
    lf.lfOrientation := 900;
    lf.lfWeight := Fw_Bold;
    StrPCopy(lf.lfFaceName, 'Arial');
    // Select this font to draw.
    OldFont := SelectObject(ACanvas.Handle,
                 CreateFontIndirect(lf));
    // Change foreground and background colors.
    clFore := SetTextColor(ACanvas.Handle, clSilver);
    clBack := SetBkColor(ACanvas.Handle, clBlack);
    // Get the menu height.
    MenuHeight := (ARect.Bottom-ARect.Top) *
      ((Sender as TMenuItem).Parent as TMenuItem).Count;
    Rectang := Rect(-1, 0, dwCheck-1, MenuHeight);
    // Draw the text.
    ExtTextOut(ACanvas.Handle, -1, MenuHeight, Eto_Clipped,
      @Rectang, 'Made in Borland', 15, nil);
    // Returns to the original state.
    DeleteObject(SelectObject(ACanvas.Handle, OldFont));
    SetTextColor(ACanvas.Handle, clFore);
    SetBkColor(ACanvas.Handle, clBack);
  end;
  // Draw the real menu text.
  ARect.Left := ARect.Left + LoWord(dwCheck) + 2;
  DrawText(ACanvas.Handle,
    PChar((Sender as TMenuItem).Caption),
    Length((Sender as TMenuItem).Caption), ARect, 0);
end;
```

Figure 6: Using *OnDrawItem* to draw vertical text on a menu.

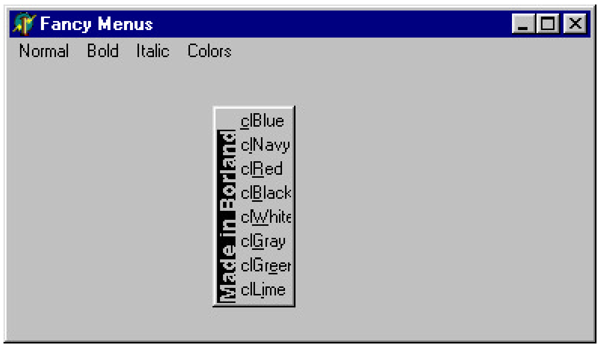


Figure 7: Menu with vertical text.

```
function CreateFont(
  nHeight,        // Logical height of font.
  nWidth,         // Logical average character width.
  nEscapement,    // Angle of escapement.
  nOrientation,   // Base-line orientation angle.
  fnWeight: Integer;   // Font weight.
  fdwItalic,           // Italic attribute flag.
  fdwUnderline,        // Underline attribute flag.
  fdwStrikeOut,        // Strikeout attribute flag.
  fdwCharSet           // Character set identifier.
  fdwOutputPrecision,  // Output precision.
  fdwClipPrecision,    // Clipping precision.
  fdwQuality,          // Output quality.
  fdwPitchAndFamily: DWORD;  // Pitch and family.
  lpszFace: PChar      // Pointer to typeface name string.
): HFONT; stdcall;
```

Figure 8: The Object Pascal declaration for the *CreateFont* Windows API function.

```
tagLOGFONTA = packed record
  lfHeight: Longint;
  lfWidth: Longint;
  lfEscapement: Longint;
  lfOrientation: Longint;
  lfWeight: Longint;
  lfItalic: Byte;
  lfUnderline: Byte;
  lfStrikeOut: Byte;
  lfCharSet: Byte;
  lfOutPrecision: Byte;
  lfClipPrecision: Byte;
  lfQuality: Byte;
  lfPitchAndFamily: Byte;
  lfFaceName: array[0..LF_FACESIZE - 1] of AnsiChar;
end;
TLogFontA = tagLOGFONTA;
TLogFont = TLogFontA;
```

Figure 9: The *TLogFont* record.

Another way to use the *OnDrawItem* and *OnMeasureItem* events is to write text vertically on a menu (as shown in Figure 7). To do this, you must create a rotated font. This is only possible using the Windows API function *CreateFont* or *CreateLogFont* (see the "Rotated Text" tip later in this article).

Then you must draw it in the *OnDrawItem* event handler. This event is fired every time a menu item is drawn, so if a menu has 20 items, it will be drawn 20 times. To make it faster, the vertical text will be drawn only when the menu item is selected (since there's is only one menu item selected at a time). Figure 6 shows how this is implemented with code, and Figure 7 shows the run-time result.

```
procedure TForm1.FormPaint(Sender: TObject);
var
  OldFont, NewFont : hFont;
  LogFont : TLogFont;
  i : Integer;
begin
  // Get handle of canvas font.
  OldFont := Canvas.Font.Handle;
  i := 0;
  // Transparent drawing.
  SetBkMode(Canvas.Handle, Transparent);
  // Fill LogFont structure with information
  // from current font.
  GetObject(OldFont, Sizeof(LogFont), @LogFont);
  // Angles range from 0 to 360.
  while i < 3600 do begin
    // Set escapement to new angle.
    LogFont.lfEscapement := i;
    // Create new font.
    NewFont := CreateFontIndirect(LogFont);
    // Select the font to draw.
    SelectObject(Canvas.Handle, NewFont);
    // Draw text at the middle of the form.
    TextOut(Canvas.Handle, ClientWidth div 2,
            ClientHeight div 2, 'Rotated Text', 21);
    // Clean up.
    DeleteObject(SelectObject(Canvas.Handle, OldFont));
    // Increment angle by 20 degrees.
    Inc(i, 200);
  end;
end;
```

Figure 10: Code to draw text rotated in 20-degree intervals.

One tricky detail is knowing where to begin drawing the text. It should begin at the bottom of the last item on the menu. To get its position, we get the height of the menu item, using:

```
ARect.Top - ARect.Bottom
```

and multiply it by the number of items in the menu:

```
(((Sender as TMenuItem).Parent as TMenuItem).Count)
```

Rotated Text

The Windows API allows you to draw text at any angle. To do this in Delphi, you must use the API function *CreateFont* or *CreateFontIndirect*. *CreateFont* is declared as shown in Figure 8.

While this function has many parameters, you will usually want to change only one or two attributes of the text. In such cases, you should use the *CreateFontIndirect* function instead. It takes only one argument — a record of type *TLogFont*, as shown in Figure 9.

Looking at this record, you'll notice its members match the parameters for the *CreateFont* function. The advantage of using this function/record combination is that you can fill the record's members with a known font using the *GetObject* API function, change the members you want, and create the new font.

To draw rotated text, the only member you must change is *lfEscapement*, which sets the text angle in tenths of degrees. So, if you want text drawn at 45 degrees, you must set *lfEscapement* to 450.

Notice that there are flags to draw italic, underline, and strikeout text, but there is no flag to draw bold text. This is done with the *lfWeight*

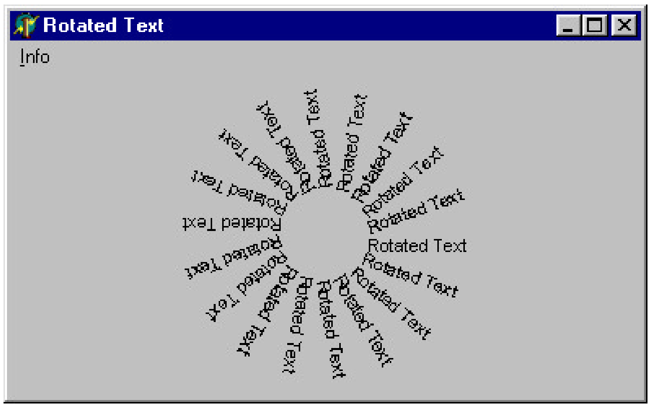


Figure 11: Text rotated 360 degrees.

```
procedure TForm1.Info1Click(Sender: TObject);
var
  LogFont : TLogFont;
begin
  // Fill LogFont structure with information
  // from current font.
  GetObject(Canvas.Font.Handle,
            Sizeof(LogFont), @LogFont);
  // Display font information.
  with LogFont do
    ShowMessage(
      'lfHeight: ' + IntToStr(lfHeight) + #13 +
      'lfWidth: ' + IntToStr(lfWidth) + #13 +
      'lfEscapement: ' +
        IntToStr(lfEscapement) + #13 +
      'lfOrientation: ' +
        IntToStr(lfOrientation) + #13 +
      'lfWeight: ' + IntToStr(lfWeight) + #13 +
      'lfItalic: ' + IntToStr(lfItalic) + #13 +
      'lfUnderline: ' +
        IntToStr(lfUnderline) + #13 +
      'lfStrikeOut: ' +
        IntToStr(lfStrikeOut) + #13 +
      'lfCharSet: ' + IntToStr(lfCharSet) + #13 +
      'lfOutPrecision: ' +
        IntToStr(lfOutPrecision) + #13 +
      'lfClipPrecision: ' +
        IntToStr(lfClipPrecision) + #13 +
      'lfQuality: ' + IntToStr(lfQuality) + #13 +
      'lfPitchAndFamily: ' +
        IntToStr(lfPitchAndFamily) + #13 +
      'lfFaceName: ' + string(lfFaceName));
end;
```

Figure 12: Getting and displaying font attributes.

member, a number between 0 and 1000. 400 is normal text, values above this draw bold text, and values below it draw light text.

The code in Figure 10 draws text at angles ranging from 0 degrees to 360 degrees, at 20-degree intervals. It's the form's *OnPaint* event handler, so the text is redrawn each time the form is painted. Figure 11 shows the result.

The form's font is set to Arial, a TrueType font. This code works only with TrueType fonts; other kinds of fonts don't support text rotation. To get current font settings and fill the *TLogFont* structure, you must use the *GetObject* API function. The code in Figure 12 shows how to fill and display the *TLogFont* settings for the form's font.

Once you have the settings in a *TLogFont* structure, the only change left is to set *lfEscapement* to the desired angle and create a new font with *CreateFontIndirect*. Before using this new font, it must be selected with *SelectObject*. Another way is to assign the handle of this new font to the handle of the canvas' font, before drawing the text. After drawing the text, this work must be reversed; the old font must be selected, and the new font deleted. If the new font isn't deleted, there will be a memory leak, and — if the routine is executed many times — Windows (especially 95/98) will run out of resources, and crash.

```
function LineDDA(
  // x-coordinate of line's starting point.
  nXStart,
  // y-coordinate of line's starting point.
  nYStart,
  // x-coordinate of line's ending point.
  nXEnd,
  // y-coordinate of line's ending point.
  YEnd : Integer;
   // Address of application-defined callback function.
  lpLineFunc : TFNLineDDAProc;
  lpData : LPARAM // Address of application-defined data.
): BOOL; stdcall;
```

Figure 13: Object Pascal declaration for the Windows API function, *LineDDA*.

```
type
  TForm1 = class(TForm)
    ImageList1: TImageList;
    procedure FormPaint(Sender: TObject);
    procedure FormResize(Sender: TObject);
  end;

var
  Form1: TForm1;

procedure CallDDA(x, y: Integer; Form: TForm1); stdcall;

implementation

{ $R *.DFM }

procedure CallDDA(x, y: Integer; Form: TForm1);
begin
  if x mod 13 = 0 then
    Form.ImageList1.Draw(Form.Canvas, x, y, 0);
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
  LineDDA(0, 0, ClientWidth, ClientHeight,
          @CallDDA, Integer(Self));
end;

procedure TForm1.FormResize(Sender: TObject);
begin
  Invalidate;
end;
```

Figure 14: Code to draw a line of bitmaps.

Stylish Lines

When you draw lines, the individual pixels usually don't matter; you simply set the line style, and it's drawn by Windows. Sometimes however, you need to do something special and draw a line style not provided by Windows. This can be done using a Windows API function named *LineDDA*, defined in Figure 13.

The first four parameters are the starting and ending points of the line. The fifth parameter is a callback function that will be called every time a pixel should be drawn. You put your drawing routines there. The last parameter is a user parameter that will be passed to

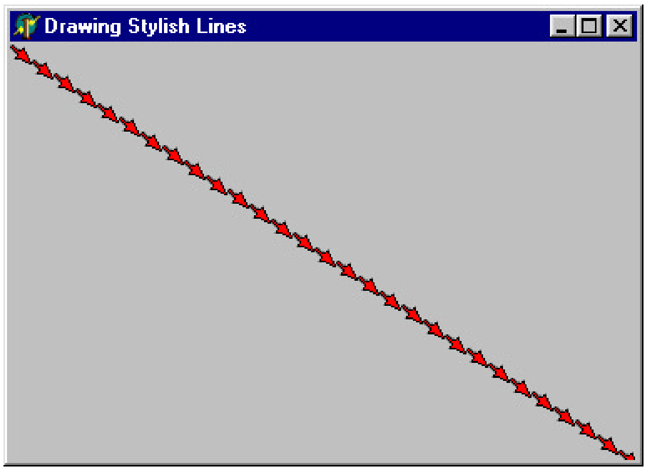


Figure 15: Window with a custom line.

the callback function. You can pass any Integer or pointer to the function, because it is an *LParam* (in Win32, it is translated to a Longint). The callback function must take the form shown here:

```
procedure CallBackDDA(x, y: Integer;
  UserParam: LParam); stdcall;
```

where *x* and *y* are the coordinates of the drawn point, and *UserParam* is a parameter that is passed to the function. This function must be declared as **stdcall**. The routine in Figure 14 draws a line of bitmaps, and Figure 15 shows the result.

This routine handles the form's *OnPaint* event, calling *LineDDA*, so every time the form must be painted, it redraws the line. Another event that is handled is *OnResize*, which invalidates the form client area, so the line must be redrawn when someone changes its size. The *LineDDA* callback function, *CallDDA*, is very simple. At every 13th point it is called, it draws the bitmap stored in the ImageList. As you may notice, *Self* is passed as the last parameter to the callback function, so it can access the instance data.

Conclusion

Since owner drawing was exposed in *TMainMenu* in Delphi 4, there have been many ways to augment your menus. Using the techniques we've discussed here, you can easily enhance your Delphi application's menus with custom text, bitmaps, and colors. ∆

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\DEC\DI200012BS.

A Brazilian, Bruno Sonnino has been developing with Delphi since its first version in 1995. He has written the books *365 Delphi Tips* and *Developing Applications in Delphi 5*, published in Portuguese. He can be reached at sonnino@netmogi.com.br.

Case Study

By Denis Perrotti

ASSESS

Delphi Helps American Skandia Compete

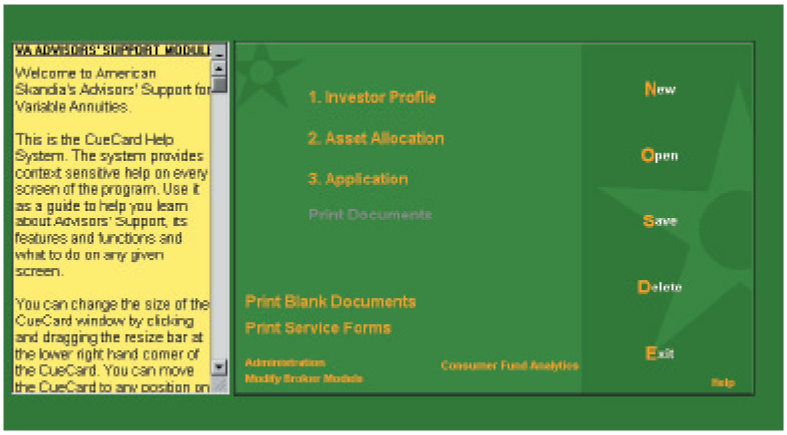
American Skandia is a relative newcomer in the world of financial services. Nevertheless, in a few years it has risen to become the number-one provider of independently sold, variable annuities in the nation. It also owns the fastest growing mutual fund complex, the American Skandia Advisor Funds. We've achieved that success despite the fact that we don't market directly to the public, or employ a captive sales force.

The lack of a captive sales force is key. We sell only through independent financial professionals who can sell anyone's products. To counter that, American Skandia offers exceptional products and provides a range of services that offer extra value to the brokers who sell those products. ASSESS® is a cornerstone of that value-added strategy.

The Low-Down on ASSESS

ASSESS is a suite of programs designed to assist financial professionals with every aspect of the sales process. The programs can track clients and their personal data, present information about the individual portfolios offered for investment in our products, and help the financial professional complete applications and other paperwork.

There's even a multimedia program packed with information about American Skandia and its products, as well as general information about investing — including more than 400MB of videos and presentations. Today, ASSESS is distributed to more than 10,000 financial professionals each quarter, and has won wide praise and recognition, including an award for top CD-ROM in the industry in 1998.



Welcome to American Skandia's Advisors' Support for Variable Annuities.

This is the CueCard Help System. The system provides context sensitive help on every screen of the program. Use it as a guide to help you learn about Advisors' Support, its features and functions and what to do on any given screen.

You can change the size of the CueCard window by clicking and dragging the resize bar at the lower right hand corner of the CueCard. You can move the CueCard to any position on

The Issues

Originally, ASSESS consisted of only two programs (the current total is 14) that supported only one product line: variable annuities. At that time it was programmed in Microsoft Visual Basic, which served us quite well. Then the company began to expand into other lines of financial products, including qualified plans, mutual funds, and variable life insurance. The challenge was clear: The programming team (which consisted of two people at the time!) was going to have to support each of those new product lines on ridiculously short deadlines.

About that same time, we needed to move away from the standard Windows look to something more representative of American Skandia. The design department set to work to create that look and was given total creative control. So they wouldn't impose limits on themselves, they were told to assume we could do anything, do it tomorrow, and do it for no money. The design they came up with was visually appealing and decidedly non-standard. This was the look American Skandia was to build.

Finally, since our installed base was about to expand tremendously, ASSESS needed to be easily deployable. We ran into numerous problems with the Visual Basic deployment, including the usual trips into DLL hell, with occasional detours into VBX purgatory. It was an experience we were anxious not to repeat.

Given these conditions, it was easy to see that Visual Basic couldn't meet our needs. Because many of the programs were going to be similar in nature, design, and structure, the development tool needed to be fully object-oriented to make the best use and re-use of code. Because we needed to create a whole new look from scratch, our new development tool needed to give us the ability to quickly and easily create new controls and components. Finally, since our new look required extensive palette manipulation given the custom color scheme, our new tool needed to be powerful and provide full access to the Windows API.

Delphi Case File

American Skandia is a leading marketer of variable annuities and mutual funds. ASSESS is a proprietary software product designed to assist financial professionals in the sale of those and other financial products. It's built on a custom architecture that allows for rapid development and customization.

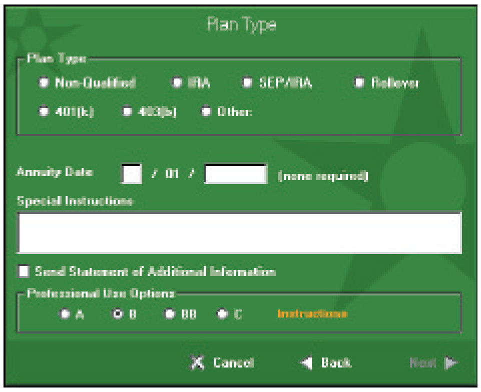
Third-party tools: Delphi, Wise Installation System, TCompress

American Skandia
One Corporate Drive
Tower One
Shelton, CT, 06484
Phone: 1-800-SKANDIA
Web Site: http://www.AmericanSkandia.com

Delphi to the Rescue

It didn't take long to see that Delphi was the only tool available that would meet all of our needs. It was fully object-oriented, with a well-defined and remarkably flexible object model. Only Java, which didn't exist at the time, exceeds the Delphi object model in terms of flexibility and power. Also, Delphi's visual environment was — and still is — second to none. Delphi's ability to produce true, stand-alone executables was also highly desirable. Because we don't control our users' machines, the fewer dependencies we have to worry about, the better. There are none with Delphi.

The ease with which we could create entirely new visual and non-visual components sealed the deal for Delphi. We created more than two dozen brand-new, customized, and highly stylized controls for use in ASSESS with remarkably little work, and this was in 1997. Visual Basic wouldn't have the ability to create controls for some time, and — even today — control creation in Visual Basic isn't in the same league with Delphi. We were also able to include advanced GUI features in our controls and programs, like pop-up buttons and mouse-overs. Such things may be common today, but they weren't three years ago.



The Aftermath

Delphi has served us well for the better part of three years. Despite ever-changing requirements and our company's "We need it now" attitude, we have continued to deliver ASSESS on time and on budget every quarter, while launching three new product lines, and the dozen new programs that support them. I can't imagine another tool that would have allowed us to do that. ∆

Denis M. Perrotti is Senior Programmer for E-Marketing, American Skandia Marketing, Inc., based in Shelton, CT. He can be reached at Dperrotti@Skandia.com.

By Bill Todd

LEADTOOLS Raster Imaging 12

A Toolkit with Intense Imaging Power

If you need a powerful professional imaging toolkit that you can use as the foundation for any graphics application, look no further than LEADTOOLS Raster Imaging 12 from LEAD Technologies, Inc. LEADTOOLS includes support for more than 60 graphics file formats, including JPEG, GIF, TIFF (including G3/G4, LZW, CMYK, and JTIF), PNG, BMP, MODCA/IOCA, PCX, and TGA.

You can manipulate all or part of an image using more than 70 digital filters and transforms, including smooth; sharpen; edge detect and enhance; change brightness, contrast, and gamma; change hue and saturation; add noise; resize and rotate images; and change color depth. All standard image compression techniques, including JPEG, CMP, LZW, G3/G4, Huffman, and run-length, are available, as is LEADTOOLS' high-performance proprietary compression algorithm. You can also apply 2,000 display effects to modify the appearance of an image.

The LEADTOOLS Family

The first challenge you'll face when evaluating LEADTOOLS is deciding which product you need. The LEADTOOLS family consists of nine products:

- Raster Imaging
- Raster Imaging Pro
- Multimedia
- Multimedia Pro
- Vector Imaging Pro
- Document Imaging
- Document Imaging Pro
- Medical Imaging
- Medical Imaging Pro

For help choosing the right product, go to the LEAD Technologies Web site at http://www.leadtools.com, click on **Products**, then click on **Comparison Chart** to see which features are included in each product. Note that although the Raster Imaging and Multimedia products don't require a royalty on each copy of your software, the document, medical, and vector imaging products do. Even if you purchase distribution licenses in advance, you must file periodic reports with LEADTOOLS indicating the number of copies deployed both within your organization and to outside users, unless you purchase a perpetual unlimited deployment license. Royalty amounts vary based on the number of units shipped. For example, if you use the document imaging product, the royalty varies from US$75 per unit for quantities less than 50, to US$1.50 per unit for quantities of 10,000 or more.

Although all of the functionality of LEADTOOLS is implemented in a collection of DLLs, you have your choice of two high-level interfaces to make programming your graphics application easier. If you're working in Delphi or C++Builder, you'll want to use the VCL components. For other languages, use the ActiveX control.

Components

The main VCL component is the LeadImage control. This visual component lets you load images from a file, the Internet, or memory, display the image on a form, and apply any of the vast collection of filters or effects that LEADTOOLS provides. You can scale the image to fit the control or view the image full size. If the image doesn't fit in the LeadImage component, you can display a pan window. The pan window, shown in Figure 1, is a thumbnail with a red rectangle that outlines the visible area of the image. Simply drag within the thumbnail to reposition the visible area. It's much easier than using scrollbars to move around. Figure 2 shows the same flower scaled to fit the LeadImage control, flipped, darkened, and with its hue adjusted to add more green.

LEADTOOLS includes a LeadTwain component, which lets you acquire images from any TWAIN device, such as a scanner or digital camera. You can display the device's user interface to let users set the acquisition parameters, or hide the device's built-in interface and design your own, passing all settings to the device through properties of the LeadImage component. The LeadIsis control provides the same image acquisition features for ISIS devices, but only functions if you have one of the document or medical versions of LEADTOOLS.

If you need to provide screen-capture capabilities in your application, simply drop a LeadScr component on a form, and you're ready to go. This component lets you capture the full screen, active window, active client area, menu under the mouse cursor, window under the mouse cursor, selected object, mouse cursor, or desktop wallpaper. If these standard screen objects don't meet your needs, you can capture any area of the screen by specifying the size and location of one of eight standard shapes, or by letting the user make a freehand selection of the area to capture. You can also capture cursor, icon, and bitmap resources from 16- and 32-bit DLL and EXE files.

The image common dialog box component provides a series of dialog boxes similar to the Windows common dialog boxes, but with added imaging features. It provides dialog boxes for File Open, File Save, all of the image processing options, and all of the image effects. The File Open dialog box provides a thumbnail preview of the image in the selected file. All of the image processing and effects dialog boxes provide a thumbnail preview that lets you see the effect before you apply it to your image. An optional **Help** button is also available in each dialog box, so you can easily integrate context-sensitive help for all of the features.

The image list component lets you display and manipulate a list of images. You can think of the image list component as a visual *TList* for images. Methods let you load images from, and save images to files, insert images, remove images, and clear the entire list. A host of properties let you control how the items in the list appear. These include everything from colors to whether the text associated with the image is displayed with the image. A set of events, including *OnClick*, *OnDblClick*, *OnItemSelected*, *OnResize*, *OnScroll*, and the mouse and keyboard events inherited from *TWinControl*, make it easy for you to let the user interact with the images in the image list in any way you wish.

To make it easy for users to select images from files, use the thumbnail browser component. This component scans a directory and generates a thumbnail image for every image file in the directory. The last component on the LEADTOOLS palette is the DICOM component, which provides DICOM file support if you have one of the medical imaging products.

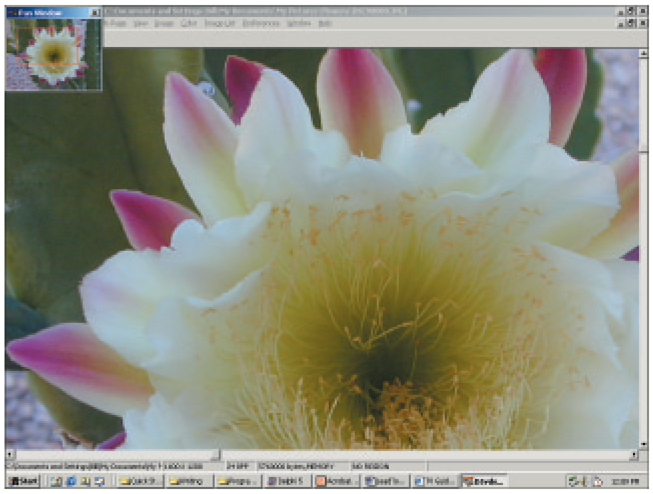


Figure 1: The LeadImage control with a pan window displayed.

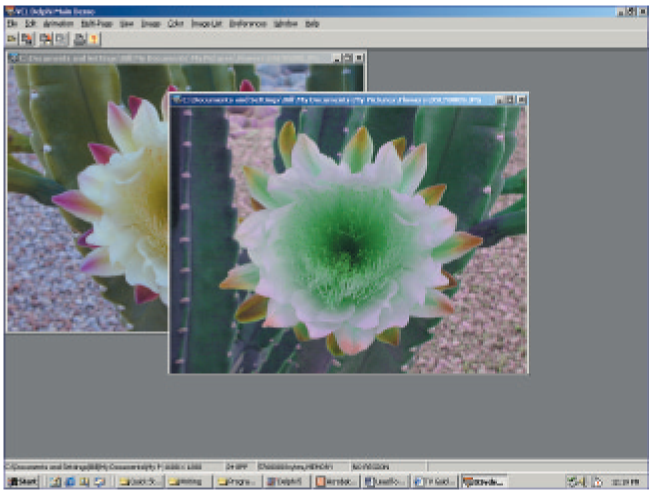


Figure 2: An image with altered hue and brightness.

Documentation

Although LEADTOOLS is a technically excellent toolkit and claims to be the world leader with an impressive list of users (including Microsoft, Hewlett-Packard, and Corel, to name a few), the documentation leaves something to be desired. The only printed documentation is a small manual that provides six pages of installation instructions and 69 pages of marketing information about the LEADTOOLS product line. Online documentation includes a 993-page manual in PDF format, and a help file that contains the same information. You can purchase the manual in printed form if you wish.

You would think that with this volume of information, everything you need to know would be there — but I didn't find that to be the case. The manual is very long on topics that tell you what LEADTOOLS can do, and very short on topics that tell you how to do it. For example, if you search the manual for Screen Capture, you will find a section titled "Implementing Screen Capture." However, the only information in that section is a description of what the screen capture component can do. There is not one word about how to do it, nor is there any reference to any other topic that describes how to capture screens.

Delphi and C++Builder developers are used to going to the help index, entering a class name, and being taken to a topic that provides a description of the class and a list of all of the properties, methods, and events for the class with each property, method, and event providing a link to its own topic. If you search either the LEADTOOLS online help or the manual for a component class name, such as *TLeadImage*, you will find nothing. There is no way to find a topic that will give you a list of the properties, methods, and events of any of the LEADTOOLS components. If you know the name of a property, method, or event, you can find its help topic, but the only way to find the names is to use the

Informant Fact File

LEADTOOLS 12 is an excellent graphics toolkit you can use as the foundation for any application that needs to acquire, manipulate, display, or print images in any common format. Although the documentation shortcomings of LEADTOOLS 12 make learning to use it more difficult than it needs to be, this is still a superior product that you should seriously consider.

Lead Technologies, Inc.
900 Baxter Street
Charlotte, NC 28204

Phone: (800) 637-4699 or (704) 332-5532
Web Site: http://www.leadtools.com
Price: US$495

Object Inspector for published properties and events; read the overview chapters of the manual, which list some of the properties, methods, and events; and browse the tutorial code and sample application.

There is a tutorial chapter for Delphi developers, and another for C++Builder developers, that steps you through building programs that demonstrate many of the features of LEADTOOLS. These chapters are a very valuable resource, but even here there are problems.

Suppose you want to acquire an image from a TWAIN scanner. You can go to the section in the Delphi tutorial titled "Creating a TWAIN Project." This tutorial consists of eight steps that tell you to drop a LeadImage and a LeadTwain component on the main form, add four lines of code to the form's *OnShow* event handler to set the values of four properties, run the program to test it, and save the project as the starting point for other tasks in the tutorial. Testing the application seems kind of silly because it's obvious that the only thing it will do is display a blank form. A natural step would be to move on to the next section to learn how to actually acquire an image from a TWAIN device. Whoops! The next section in the tutorial is on creating, viewing, and merging color separations. Now what? Let's go back to the table of contents and find the next section in the tutorial that deals with TWAIN. Whoops! There are no other sections that mention the word TWAIN. Fortunately, because the manual is in PDF format, you can do a global search for TWAIN and eventually, you'll find that the rest of the TWAIN example is buried in the section named "Miscellaneous Examples" at the end of the tutorial chapter.

There is also an extensive demonstration application that shows how to use most of the features of LEADTOOLS. Versions of the demonstration application are available for Delphi 4 and 5 and for C++Builder 4 and 5. Unfortunately, there are problems here as well. When you run the Delphi 5 version of the demonstration and choose File | Browse from the menu, all you get is an error dialog box with the message "GetDirectory: invalid parameter passed."

Conclusion

LEADTOOLS is an excellent graphics toolkit that you can use as the foundation for any application that needs to acquire, manipulate, display, or print images in any common format. You cannot have this much power without complexity and, as a result, the LEADTOOLS components include a very large number of properties, methods, and events that you'll need to master. Although the documentation shortcomings make learning to use the toolkit more difficult than it needs to be, this is still a superior product that you should seriously consider if you need to do high-end image processing. Δ

Bill Todd is president of The Database Group, Inc., a database consulting and development firm based near Phoenix. He is co-author of four database programming books, author of more than 60 articles, a Contributing Editor to *Delphi Informant Magazine*, and a member of Team Borland, providing technical support on the Borland Internet newsgroups. He is a frequent speaker at Borland Developer Conferences in the US and Europe. Bill is also a nationally known trainer and has taught Delphi programming classes across the country and overseas. He is currently a speaker on the Delphi Development Seminars Kylix World Tour. Bill can be reached at bill@dbginc.com. For more information on the Kylix World Tour, visit http://www.DelphiDevelopmentSeminars.com.

Wireless Web Development

Interest in wireless application development is exploding. Nearly every major portal, content provider, and eBusiness site is evolving some form of wireless support for their services. Given the current state of health in the wireless communications market, it's not surprising, especially since this is the first time in history that so much information can be obtained through such a small, portable communications device.

Naturally, the servers providing content to desktop browsers have to be instructed how to handle this new era of mobile information interaction. And like the amazing effort that went into the construction of the Internet, many hard working developers will have to rise to the challenge and expend the mental effort to understand, practice, and instruct others in wireless application development best practices.

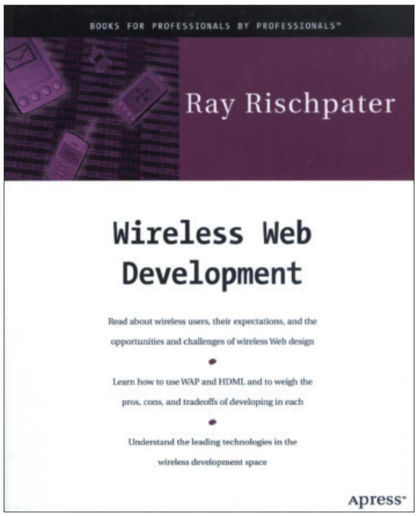
To that end, Ray Rischpater provides a primer to the wide world of the wireless Web in his *Wireless Web Development*. Although the primary intended audience is existing Web professionals, Ray's coverage of topics run the gamut from the rudiments of HTML to specific server-side scripting technologies such as the open-sourced PHP language. Given that the book is only 350 pages, covering such a broad spectrum of information is tricky. And while Ray's mad dash through the various flavors of both proprietary and open wireless standards is commendable, the usefulness of the book to the trained Web professional is diluted and less valuable as a result.

On the positive side, *Wireless Web Development* provides a fairly comprehensive, albeit slightly dated, summary of the various wireless infrastructures, protocols, and markup languages available for the various wireless and offline clients. Ray summarizes in efficient prose some of the more commercially visible markup variants, including Palm's Web Clipping Architecture, AvantGo's channel service, Microsoft's Channel Definition Format for mobile devices (i.e. Windows CE/Pocket PC clients), Phone.com's Handheld Device Markup Language (HDML), and WAP Forum's Wireless Markup Language (WML). Additionally, Ray devotes a chapter to the emerging, but sparsely implemented, WMLScripting client-side language that's beginning to show up in high-end WAP-enabled mobile devices.

I also enjoyed Ray's conversational writing style. Although this book is intended for techies, Ray keeps the tone light and general enough for an average reader to finish the book in a day. As indicated earlier, however, this generalization of the subject matter also detracts from the book's value.

Unfortunately, my list of negative aspects regarding *Wireless Web Development* exceeds its praises. I'm fairly adept at wireless application development, which is one of the reasons I was asked to review this title. This background altered my expectations; I was anticipating a much deeper discussion of wireless Web development.

The book blazes through dynamic server-side scripting, and the author exhibits a bias toward PHP as the server script of choice. Granted, this free, cross-platform, open source technology is a worthwhile contender in dynamic wireless content generation, but I would have preferred to also


BOOKS FOR PROFESSIONALS BY PROFESSIONALS™

Ray Rischpater

Wireless Web Development

Read about wireless users, their expectations, and the opportunities and challenges of wireless Web design

•

Learn how to use WAP and HDML and to weigh the pros, cons, and tradeoffs of developing in each

•

Understand the leading technologies in the wireless development space

Apress®

see examples in ASP, JSP, CFML, and Perl/CGI, which are more prevalent technologies from an ISP-hosting point of view. Additionally, there are many more books published on these languages than there are on PHP. At the very least, I would have preferred to see an area dedicated to source variants on Apress' Web site along with the book's printed code listings.

On the topic of references, I was also disappointed that Ray omitted a list of free tools and services available to wireless Web developers. In my own exploration, I found Inetis' DotWAP (available at http://www.inetis.com/english/solutions_dotwap.htm) to be a great time saver when generating basic WML cards. Additionally, WAPJAG (http://www.wapjag.com) provides a free Phone.com-like WML-compliant client to view a broad range of Internet-accessible WML content. Lastly, I really like Chami's excellent and free HTML-Kit (available at http://download.cnet.com/downloads/0-10070-100-1507274.html?tag=st.dl.10001_103_1.lst.td), which offers a commercial-grade Web scripting editor with full support for nearly every major server and client-side Web language, including Ray's preference, PHP. Given that the book doesn't include a CD-ROM, the least it could provide is a more comprehensive list of tool resources.

Another unfortunate omission — one that would have been highly beneficial as an appendix — is resolving the aggravating question of how a wireless Web developer can effectively support the many flavors of client-side scripts and presentation layers introduced in the book. As it stands, an untrained developer might approach the problem by manually creating multiple site sections for each platform the developer wishes to support. Instead, even a cursory introduction to an effectively designed *n*-tier architecture separating data, business, and presentation logic would have communicated a simple, yet powerful message: A well-designed foundation is crucial to the success of any Web site's scalability and support for the ever-evolving wireless Web.

Finally, continuing the discussion of omissions, I would have preferred to see a keyword summary listing in separate appendices for each of the scripting technologies introduced in the book. As it stands, developers expecting to use the title as a reference will find it frustrating to have to leaf through chapter pages to locate a single tag's meaning.

As with most titles written by working professionals today, it's obvious that the author spent many evenings and weekends of his valuable time to put into words his understanding of the subject matter. I have no doubt that Ray is a talented Web developer who struggled with what to discuss and what to leave out. On a positive note, I was quite pleased that Ray's book is one of the few general Web development books that intelligently advocates the use of Unified Modeling Language (UML). All of the sample program flows are illustrated throughout the book in easy to understand UML diagrams, and Ray even dedicates a full appendix to the language. Regrettably, there simply aren't enough of these appendices in the book to make it a regular reference for wireless Web developers.

— *Mike Riley*

Wireless Web Development by Ray Rischpater, Apress, 901 Grayson St., Berkeley, CA 94710-2617, http://www.apress.com.

ISBN: 1-893115-20-8
Price: US$34.95 (350 pages)

Advanced Delphi Developer's Guide to ADO

When I met Dr Natalia Elmanova (one of the authors) at last summer's Inprise/Borland Conference, she asked me immediately if I would review this book. I didn't know of it at the time, so I gave her a tentative "yes." I should mention that I am quite selective about the books and products I review; I don't enjoy slamming a work, or wasting my time for that matter. When my copy of *Advanced Delphi Developer's Guide to ADO* arrived, I realized I had no choice; I had to review this book. Let's find out why.

Providing a solid background for working with ADO. The first observation I made while reading the opening chapters was this: The authors, Dr Elmanova and Alex Fedorov, were real educators. Not only do they have a solid grasp of the subject matter, they're able to present it in a very clear manner. To their credit, they leave nothing to chance, making few assumptions about the potential reader's level of knowledge or experience. I concur with them that this book will be accessible and valuable to those "programmers who are already familiar with Delphi, but are novices in using ADO." In fact, the book assumes little or no knowledge about database programming, and devotes the opening chapter to one of the most cogent introductions to this topic I have ever seen.

As you may know, ADO (Microsoft ActiveX Data Objects) is one of several technologies that enable universal data access. Fortunately for us, it's a technology that Delphi supports. In the second chapter, the authors discuss ADO in its larger context, giving the reader an excellent overview of what is available. Chapter 3 discusses another of these technologies, OLE DB, and provides the first code examples. Chapter 4 returns to more basic issues, discussing the Delphi Database Architecture and the components that support it.

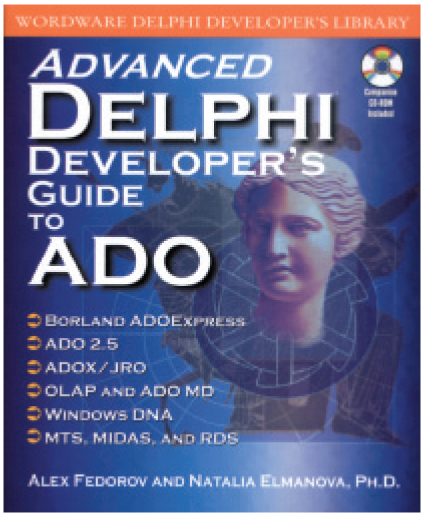
The next three chapters are devoted to ADO, exploring the basic components *TADOConnection*, *TADOCommand*, and *TADODataSet*. The next chapter exposes three more ADO components that have parallels in Delphi's standard database components: *TADOTable*, *TADOQuery*, and *TADOStoredProc*. No database book would be complete without a chapter on SQL, and *Advanced Delphi Developer's Guide to ADO* includes a fine one. The first half of the book ends with two chapters, "Working with Database Objects" and "Building Database Applications."

Exploring advanced ADO topics. Having laid a solid foundation in the first half, the second half explores more advanced topics, such as working with business graphics, reports, and analysis systems using ADO and Delphi components. Unfortunately, not every aspect of, or extension to, ADO is covered by Delphi components; the authors address this issue in some of the remaining chapters. Chapter 15 and those that follow are longer and introduce more difficult topics. Chapter 15, for example, shows how to use type libraries to access ADO MD (multidimensional) objects. Chapter 16 explains ADO DLL and security extensions; Chapter 17 covers Microsoft's Jet and Replication library; and Chapter 18 explains the somewhat complicated issues of deploying ADO applications built with Delphi.

The remaining chapters deal mostly with issues surrounding distributed applications using Microsoft's Remote Data service, Borland's MIDAS, and the Microsoft Transaction Server. The final chapter introduces the latest version of ADO (2.5), which is an integral part of Windows 2000. Before closing, I should point out an area or two of weakness and alert you to some issues in running the code.

One area that could be improved: running the code. The weaknesses in this book are quite minimal. While the index is good, it should be more detailed. For example, at various places in the book, the authors make reference to Windows 2000, but there is no reference to Windows 2000 in the index.

Another area I should warn you about concerns running the applications. Of course most readers who copy Delphi code from a CD to their hard drive are already aware of the need to change the read-only property of those files.



Another thing you'll need to do is un-check the Build with Run-time packages check box in the Project Options dialog box. You'll probably already have an appropriate database installed on your system; if not, you'll need to take care of that as well. Some good news: An updated version of the code is available already at http://d5ado.homepage.com.

To conclude, this is a most impressive work — an excellent, well organized, and well written treatise. The authors assume nothing in terms of the reader's background in database programming or ADO. They provide you with all of the background information you might require, and build on that foundation by providing an excellent exposition of ADO, including some important advanced topics. For any reader who will be creating Delphi applications that use this important technology, I recommend this book highly. It's an essential resource.

— *Alan C. Moore, Ph.D.*

Advanced Delphi Developer's Guide to ADO by Natalia Elmanova, Ph.D. and Alex Fedorov, Wordware Publishing, Inc., 2320 Los Rios Blvd., Plano, TX 75074, http://www.wordware.com.

ISBN: 1-55622-758-2

All Developers Are in R&D

All developers are in R&D — or should be. Personal research (books, magazines, newsgroups, Web sites, and source code) and development (hacking and side projects) are necessary if you're to grow as a developer and adapt to changing needs. In contrast, developers who concentrate only on their assigned tasks often harm themselves and their employers.

It's self-defeating to subscribe to the "begin coding as soon as possible to finish the project as soon as possible" mindset. It's specious reasoning. Although it sounds sensible on the surface, it often leads to chaos and project cancellation. It's the same as thinking that devoting all your time to a project (and thereby neglecting research and development) is selfless and valorous. Such self-sacrifice is deleterious to you and your employer or client. Not opening up to better ways means you're not doing your best work.

Failing to aerate the brain and rejuvenate the creative juices leads to an atrophy of spirit and an inbreeding of thought. To quote Charlie Calvert on the importance of hacking from *Delphi Unleashed*: "Most good programmers spend a large percentage of their time hacking. If you run a programming shop and you arrange things so that none of your programmers have time to hack, you will end up with a group of very mediocre programmers. The best will either lose their skills, or more likely, head for greener pastures."

If we rely only on personal prior knowledge (been there, done that) and hard work (I don't have time to stop and sharpen the saw, I'll just work harder/faster), we are limiting our productivity.

What you don't know *can* hurt you. An example of this is my first program in Delphi 1. Its sole purpose was to return the day of the week for a date entered by the user. I labored over this little application, and wrote so much complex code, that I got a run-time error for using up too much stack space. Finally, the happy moment came when the code was broken up into smaller methods, the stack space was increased, and the program was debugged. It worked like a charm. No papa was prouder as I showed off my baby.

Not much later, though, I discovered the standard Delphi *DayOfWeek* function. I could have saved myself hours using this function and writing just one line of code! To punish myself I re-wrote the application from scratch, using the preferred method. I finished in approximately 15 minutes. Yes, what you don't know can come back to chomp you on the gluteus maximus!

When you get a new version of Delphi, do you simply kick the tires and take it for a spin, or do you also look under the hood? As Danny Thorpe said: "Use the source, Luke!" Read it. Seek out the changes from the last version. What do the new components do? What classes have been added that don't show up on the Component palette? What functions are new to this version? It would be a shame to rewrite code that has already been written, tested, and debugged for you. In Delphi 6 take a look at the new declaration of *TComponent*, the new utilities in the Math unit, new units such as StrUtils, ConvUtils, DateUtils, VarUtils, Bands, etc. Check out the *IfThen*, *AnsiIndexText*, *Soundex*, and *Metaphone* functions. You won't know they're available if you don't look for them. Be curious; become a Delphi expert!

Expand your knowledge: Pick your colleagues' brains, explore, and experiment. Reading, sharing with others, and scouring OPC (other people's code), should give you countless ideas for utilities. Side projects and hacking, and using technology or components different from your usual group will expand your horizons. Write a database application using a different DBMS and/or engine. Write components that answer tricky programming challenges or Delphi FAQs, so you can say: "Just use this component; the functionality you're after is automatically provided." The possibilities for personal advancement and self-expression are virtually endless. Grab them before they grab you. Innovate or stagnate! Δ

— *Clay Shannon*

Clay Shannon is an independent Delphi consultant based in northern Idaho. He is available for Delphi consulting work in the greater Spokane/Coeur d'Alene areas, remote development (no job too small!), and short-term or part-time assignments in other locales. Clay is a certified Delphi 5 developer, and is the author of *Developer's Guide to Delphi Troubleshooting* [Wordware, 1999]. You can reach him at bclayshannon@earthlink.net.

The Delphi Toolbox: In the Bin

I wrote the first in this series of columns a year or so ago to concentrate on third-party tools, components, and libraries, providing mini reviews along with tips and techniques. Recently I realized that there are a number of hidden gems — useful but little-known utilities — that come with Delphi. All of these can be found in the Delphi\Bin folder, and many go back to early versions of Turbo Pascal.

Command-line utilities. To learn the syntax and options for command-line utilities, you can usually either type the name alone, or the name with a question mark (?) at the command prompt. One of the veterans of Turbo Pascal is TPC.EXE, the command-line compiler, with its new name DCC32.EXE. Its long list of available switches provides you with the many compilation options available in the Delphi IDE. You may be wondering, "Why use such a cumbersome tool when you have the IDE available and can simply click with the mouse?" In the old days, one motivation was memory — more was available at the command prompt than in an IDE. That's not such an issue in compiling large programs today. Another reason for using this fast compiler is to automate compilation of multiple units in a batch or "make" file, a technique used by third parties when they distribute patches to their libraries.

Let's make it. Another venerable utility, MAKE.EXE, is a program manager for compiling programs with specific options. Those options are listed in a text file that generally has a *.mak extension.

Make files, which are similar to batch files, can be used to automate many processes: from building large complex projects, to compiling resource files. Delphi comes with a large make file called, of all things, Makefile! If you have Turbo Assembler and other required files, you can run this with a recent version of MAKE.EXE to rebuild the entire VCL from the command line. You can also edit the file to create either a debug or non-debug version of the library. On the command line you can specify options (including the make file to use), and one or more target files.

Other options include specifying whether to conduct auto-dependency checks, providing the name of an include directory, and indicating whether to ignore encountered errors. You can do a lot in the main make file, such as defining macros. These can serve as shortcuts to executable files (usually compilers), including their full path and options. You can also define explicit rules and implicit rules. Implicit rules are generalizations of the explicit rules and help to simplify a make file.

Get a grep or dump it. What's this grep thing all about? Consider this scenario: Months ago you wrote a wonderful utility that accomplished its task perfectly, but now you've forgotten which programs it's in. You do remember the name of the routine, so it's grep.exe to the rescue! This fast file-searching utility provides a plethora of options, including word search, regular expression search, inclusion or exclusion character set searches, and more. Best of all, it's freely available with every version of Delphi.

TDump.exe is well named. This utility will literally dump a ton of information about an executable file to the screen. It lists the DOS file size, load-image size, header size, minimum and maximum memory requirement, program entry point, CPU type, O/S version, various flags and offsets, code and data sizes, and base locations. It provides information about exports and imports, resources, object tables, imports from Windows DLLs, and much more.

One final command-line tool might be helpful in certain circumstances. Convert.exe provides a quick and easy way to convert one or more Delphi form files to either text or binary formats. You have the option of converting the file in-place, overwriting the input file.

Wizards. In addition to the command-line tools we've been discussing, there are many useful Wizards in the \Bin subdirectory. Some are integrated into the Delphi IDE; others can be easily added to the Tools menu. There are several tools for managing databases (not available in all editions), including the BDE Administrator, which lets you configure the Borland Database Engine (BDE) among other tasks; Database Explorer, for browsing and editing database objects; and tools for working with SQL. The Image Editor is a useful tool that provides an easy way to work with graphic files you use in your applications, e.g. icons, cursors, and bitmaps. Finally, WinSight is a debugging tool that provides information about window classes, windows, and messages. For more information on these and other tools, look under Delphi Productivity Tools in the Delphi Help file.

Remember, before you go looking for a solution from a third-party source, check out what's available with Delphi. You just might discover the solution to your problem "in the bin." Until next time... ∆

— *Alan C. Moore, Ph.D.*

Alan Moore is a Professor of Music at Kentucky State University, specializing in music composition and music theory. He has been developing education-related applications with the Borland languages for more than 15 years. He is the author of *The Tomes of Delphi: Win 32 Multimedia API* [Wordware Publishing, 2000] and co-author (with John Penman) of an upcoming book in the *Tomes* series on Communications APIs. He has also published a number of articles in various technical journals. Using Delphi, he specializes in writing custom components and implementing multimedia capabilities in applications, particularly sound and music. You can reach Alan on the Internet at acmdoc@aol.com.