April 2000, Volume 6, Number 4

Developing WML Applications for Mobile Phones

Cover Art By: *Arthur Dugoni*

MicroEdge Announces Visual SlickEdit 5.0 for Windows

MicroEdge, Inc. announced the release of ***Visual SlickEdit 5.0 for Windows*** (Windows 95/98/NT), a multi-platform editing solution supporting integration of all major IDEs across all languages.

New features for Visual SlickEdit 5.0 include Context Tagging, which performs expression type, scope, and inheritance analysis on your source code, even as you type. Supported languages include Delphi, HTML, Ada, Cobol and OO Cobol, C/C++, Java, JavaScript, Perl, PV-Wave, InstallScript, and Visual SlickEdit's own Slick-C. With Visual SlickEdit 5.0, Context Tagging's Auto List Members feature will support viewing comments for symbols with the same name, and the Auto Function Help feature will display function comments along with the prototype and current argument. Context Tagging now supports C-style preprocessing for Java (Visual J++ support), and Javadoc comments are now displayed in a built-in HTML browser with hyperlink support.

The new version also offers symbol references and uses. New functionality for references include a new References tab on the Output toolbar, next/-previous reference hot keys, and a context menu item for querying references for the symbol at the cursor.

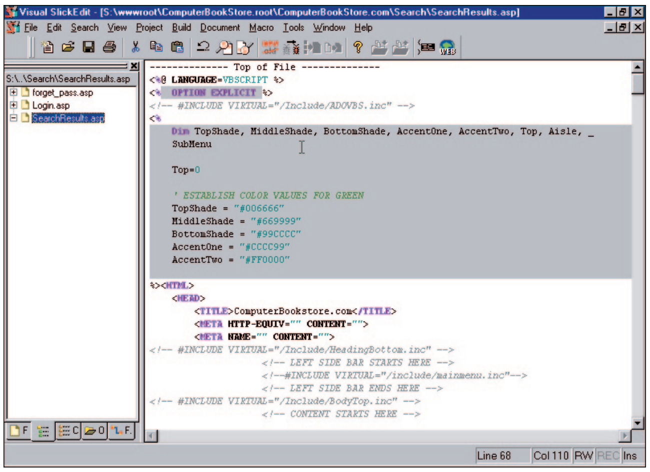
Visual SlickEdit's DIFFzilla has also been enhanced. Auto Reload now provides the option to diff an open file with the copy on disk when Visual SlickEdit detects that another application has modified the file.

Project management enhancements are also included in Visual SlickEdit 5.0. Multiple projects may now be defined in a workspace, and projects can be shared between other workspaces. Dependencies may be defined between projects in a single workspace, allowing a more sophisticated build process. Another project management enhancement is support for projects with multiple language file types.

The FTP Client toolbar and FTP Open tab now allow recursive FTP directory operations. Developers will be able to upload, download, and delete entire directories. More host support is available in Visual SlickEdit 5.0, including OS/400, VM, VOS, Windows NT, OS/2, MVS, VMS, Netware, and MacOS.

Visual SlickEdit 5.0 for Windows also provides the Javadoc Editor (supports Java, C, C++ and Slick-C), HTML and JavaScript Beautifier, emulation for Visual C++, and Print Preview/-Schemes. It also provides additional language support for PL/I, JCL, OS/390 Assembler and IDL, as well as embedded language support for JavaServer.

MicroEdge, Inc.
Price: US$295
Phone: (800) 934-EDIT
Web Site: http://www.slickedit.com


Visual SlickEdit - [S:\wwwroot\ComputerBookStore.root\ComputerBookStore.com\Search\SearchResults.asp]
File Edit Search View Project Build Document Macro Tools Window Help
S:\..\Search\SearchResults.asp
forget_pass.asp
Login.asp
SearchResults.asp
Line 68
Col 110
RW
REC
Ins

```
-------------- Top of File --------------
<%@ LANGUAGE=VBSCRIPT %>
<%  OPTION EXPLICIT %>
<!-- #INCLUDE VIRTUAL="/Include/ADOVBS.inc" -->
<%
    Dim TopShade, MiddleShade, BottomShade, AccentOne, AccentTwo, Top, Aisle, _
    SubMenu

    Top=0

    ' ESTABLISH COLOR VALUES FOR GREEN
    TopShade = "#006666"
    MiddleShade = "#669999"
    BottomShade = "#99CCCC"
    AccentOne = "#CCCC99"
    AccentTwo = "#FF0000"

%><HTML>
    <HEAD>
        <TITLE>ComputerBookstore.com</TITLE>
        <META HTTP-EQUIV="" CONTENT="">
        <META NAME="" CONTENT="">
<!-- #INCLUDE VIRTUAL="/Include/HeadingBottom.inc" -->
                    <!-- LEFT SIDE BAR STARTS HERE -->
                    <!--#INCLUDE VIRTUAL="/include/mainmenu.inc"-->
                    <!-- LEFT SIDE BAR ENDS HERE -->
<!-- #INCLUDE VIRTUAL="/Include/BodyTop.inc" -->
                    <!-- CONTENT STARTS HERE -->
```

Delphi COM Programming
Eric Harmon
Macmillan Technical Publishing

ISBN: 1-57870-221-6
Price: US$45 (510 pages)
Web Site: http://www.newriders.com

UCalc Announces UCalc Fast Math Parser 2.0

UCalc Software announced the release of ***UCalc Fast Math Parser 2.0***. This new version, which is faster and slimmer, includes direct support for Delphi, C++Builder, Visual C++, PowerBASIC, and Visual Basic. Features such as function callbacks, unlimited definition space, string support, preparsing, function aliasing, customizable separators for international users, extended precision, and more were also added.

A math parser is a tool that allows programs to evaluate algebraic expressions that are defined at run time. This is something that is frequently requested in various Usenet programming newsgroups. Without a parser, a program would be limited to pre-defined formulas. Whenever a new formula is added to a program, it would have to be recompiled. The end user of such a program would be stuck with whatever built-in formulas are available. UCalc Fast Math Parser makes it possible for programs to support dynamic manipulation of math and string expressions by the end user.

This parser is designed in such a way that third-party add-on products can easily be created. It is in a position to spawn new products, such as standard libraries of financial, scientific, and other miscellaneous functions, as well as spreadsheet-type components, with UCalc Fast Math Parser serving as the underlying number-crunching engine.

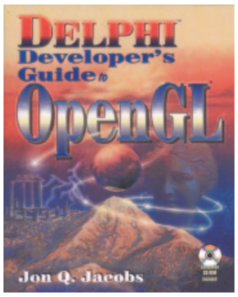
This new version is no longer an ActiveX DLL. However, not only is the new DLL faster, but the distribution of bulky run-time files is no longer required.

UCalc Software
Price: US$300 for a standard license.
Phone: (305) 233-2604
Web Site: http://www.ucalc.com/mathparser

Delphi
TOOLS

New Products and Solutions

Delphi Developer's Guide to OpenGL
Jon Q. Jacobs
Wordware Publishing, Inc.



ISBN: 1-55622-657-8
Price: US$49.95 (465 pages, CD-ROM)
Web Site: http://www.wordware.com

combit Releases List & Label 6.0

combit GmbH announced the release of *List & Label 6.0*, a new version of the company's database-independent development tool for extensive report, label, and form output functions. The real data preview can be easily integrated in intranet/Internet applications.

List & Label is available in English and German. Language kits are available for the designer, arming it in many different languages for the end user.

List & Label is available for all DLL-capable programming languages. Special versions that work with Delphi (VCL) or Visual Basic (OCX) are also available.

List & Label 6.0 consists of a print engine and a form designer. The data that should be available for use in the designer is transferred by the print engine, independent of a specific database. Virtual variables can also be used. Hierarchal variables help to maintain a good overview when working with various linked tables of data. Many code programming examples are provided for a range of programming environments to help simplify the programming process.

The DTP form designer offers encompassing layout tools, and many filter and layout options. These can be used via drag-and-drop. Special text and list objects cover many creation needs. The integrated formula assistant enables you to undertake complex calculations and string manipulations directly at run time. The integration of RTF text, graphics, and barcodes is also possible, as well as individual printer control for the first and following pages.

The programmable real data preview is additionally available as an ActiveX control, as well as in a separate EXE file. This allows the use of a List & Label preview directly in an Internet browser. You can see all information, or you can print, save, or send it as an e-mail. The ActiveX control can also be integrated into your own Internet applications. With the compact EXE viewer, you can view preview files received per e-mail, or print them when needed.

combit GmbH
Price: Call for pricing.
Phone: +49 7531 90 60 10
Web Site: http://www.combit.com

AutomatedQA Announces QTime

AutomatedQA Inc. announced the release of *QTime*, its comprehensive application testing, debugging, profiling, and coverage analysis tool for Borland Delphi, C++Builder, and Microsoft Visual C++.

QTime is specifically designed for software developers and their organizations to help deliver robust, bug-free, and bulletproof code, while simultaneously reducing the amount of time spent testing and managing the application delivery process.

Once an application is created, QTime can help guarantee that code will perform quickly with stability and overall application efficiency.

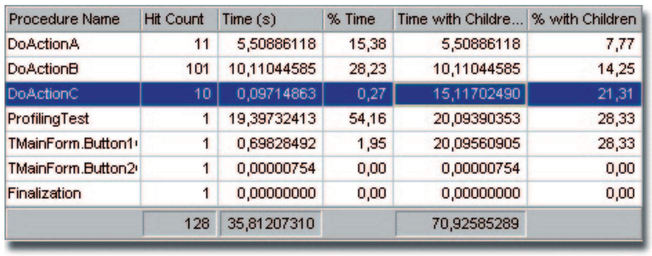
QTime includes an array of profilers to help identify specific problems, including poorly performing application algorithms (Timing Profiler); global (Sampling Profiler) and low-level (Hierarchical Profiler) application performance issues; unnecessary function calls or overly used functions calls (Hit Count Profiler); untested sections of application code and functionality (Coverage Profiler); the trace and flow of code (Trace Profiler); class usage and its impact on the long-term viability of the code (Class Review); COM reference leaks (Reference Count Profiler); and more.

QTime can give the answers to common development questions, such as:

- What is the slowest area of my program?
- What is the most used and/or executed procedure?
- Which piece of code never gets executed during a given test sequence?
- What is the slowest procedure in my program?
- What is the execution flow of my program?
- Have code changes improved application performance?
- Which units and or files are used by my program?
- Am I linking to my program units and or files that are not really used?
- Which procedures are linked to my program?
- How many procedures are used by my program?
- How many files and or units are used by my program?
- Where in the memory address space are my procedures loaded?
- What is the longest procedure (in procedure source code lines)?
- What is the biggest procedure (in bytes)?
- What is the biggest unit and or file in compiled bytes?
- Which unit contains the greatest and or lowest number of procedures?
- What is the most used class in my program?
- Do I free all classes allocated in my program?
- What is the binary output produced by the compiler for my source code?

QTime offers a totally open plug-in architecture so that others within the software development community can participate in enhancing its feature set.

AutomatedQA Inc.
Price: US$349.99
Phone: (702) 262-0609
Web Site: http://www.totalqa.com



Procedure Name	Hit Count	Time (s)	% Time	Time with Childre...	% with Children
DoActionA	11	5,50886118	15,38	5,50886118	7,77
DoActionB	101	10,11044585	28,23	10,11044585	14,25
DoActionC	10	0,09714863	0,27	15,11702490	21,31
ProfilingTest	1	19,39732413	54,16	20,09390353	28,33
TMainForm.Button1	1	0,69828492	1,95	20,09560905	28,33
TMainForm.Button2	1	0,00000754	0,00	0,00000754	0,00
Finalization	1	0,00000000	0,00	0,00000000	0,00
	128	35,81207310		70,92585289	

Object River Announces COM Express 1.0

Object River Information Technology Inc. announced ***COM Express 1.0 for Delphi with RoseLink***, a component-based *n*-tier Internet/intranet application development tool.

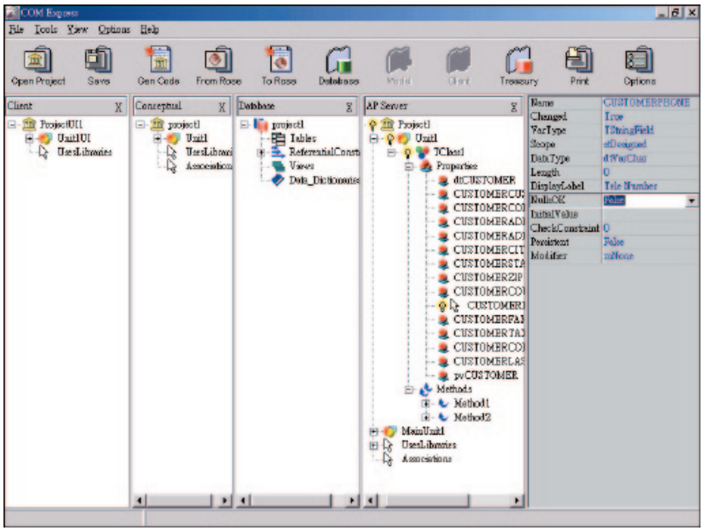
COM Express supports all tier development of business-critical Internet/intranet application throughout the life-cycle. It enables the designer to maintain database specifications and create or restructure physical tables from Oracle, MSSQL, Sybase, Informix, DB2, InterBase, etc.

With COM Express, developers can create and design business objects using simple steps, and can choose COM, MTS, COM+ Server with MIDAS, ADO, or other solutions.

It also provides an ASP page, Internet Express, or a Delphi thin client form for a presentation tier. All specifications can be generated into a Word document and customized using scripts. The UML models designed by Rational Rose can be imported, and COM Express specifications can be exported to Rational Rose.

All existing client/server Delphi projects can be converted into *n*-tier architecture automatically, and customized via drag-and-drop.

Object River Information Technology Inc.
Price: US$2,499 per set.
Phone: 886-2-87805563
Web Site: http://www.objectriver.com



Name	CUSTOMERPHONE
Changed	True
VarType	TStringField
Scope	stDesigned
DataType	dtVarChar
Length	0
DisplayLabel	Tele Number
NullsOK	False
InitialValue	
CheckConstraint	0
Persistent	False
Modifier	mNone

Fenestra Technologies Offers Event Journal through Component Factory

Fenestra Technologies Corp. announced the release of ***Event Journal***, the first product from Component Factory, Fenestra's software component e-commerce Web site.

Event Journal provides a simple, non-invasive, and automatic way to hook into and log all user-interface events (such as selecting menus, clicking buttons, and minimizing windows) for any Delphi or C++Builder application. Using Event Journal is like peering over a user's shoulder as they use an application.

By tracing events that lead up to an exception, Event Journal facilitates quality assurance and debugging. It also promotes usability by showing how an application is actually being used in the field, so developers can improve and simplify the user interface. Event Journal shows the exact sequence of events, which provides a better understanding of how an application works.

Event Journal includes events covering all the components that ship with Delphi. Developers can register their own custom events with the system, as well.

Event Journal also includes the Event Viewer for browsing, filtering, exporting, and printing an event journal log, plus documentation and examples.

Fenestra Technologies Corp.
Price: US$99
Phone: (301) 721-3912
Web Site: http://www.componentfactory.com

Extended Systems Announces Advantage Database Server 5.6

Extended Systems, Inc. announced the release of ***Advantage Database Server 5.6***, the newest version of the company's client/server DBMS for shared, networked, stand-alone, and Internet database applications.

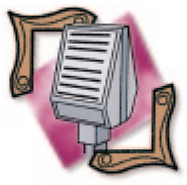
Advantage Database Server 5.6 includes Delphi source code with the Advantage Database Engine version 2.6 for Delphi. New features provide greater performance and added functionality for developers building applications with Borland Delphi, as well as various other database programming languages.

With this release, Extended Systems also added features that provide greater functionality for developers, such as SQL UNION support, sharing instances of components across threads, and support for NetWare 5 IP and SMP.

Further enhancements have been made to existing features as well, such as JOIN optimization and ORDER BY on aggregates, which increase the performance of Advantage Database Server.

Extended Systems, Inc.
Price: From US$615 for the five-user, Windows NT and NetWare versions, to US$7,495 for the unlimited version; client kit prices range from US$99 to US$299.
Phone: (800) 235-7576 x5030
Web Site: http://www.AdvantageDatabase.com

table of CONTENTS



Baltic Solutions Provides Web Site for Products Built with Delphi

Klaipeda, Lithuania — Baltic Solutions is offering a new service for the Delphi community: a Web site featuring software products built with Delphi. The site can be found at http://www.balticsolutions.com/bwd.

The purpose of this site is to provide a consolidated list of all quality commercial, shareware, and freeware products that have been developed using Borland Delphi.

Anyone with a product they would like added to the database can simply submit it to Baltic Solutions, and it will be added to the listings.

Currently, it lists only Delphi-built products, but Baltic Solutions is planning to expand it to hold software built with C++Builder, JBuilder, and other Borland tools.

As of January 11, 2000, the database contains 175 products, most of which are available on the Internet. The company encourages people to submit even in-house products, modules of larger IT systems, etc., as long as they are completed and can serve as success stories of Delphi use.

DelphiMag.com Receives Best of the Net Award

New York, NY— DelphiMag.com was selected as one of the top Delphi programming-related sites on the Internet by About.com's Delphi Programming site.

This is the first-ever About.com Best of the Net Award for a Delphi-related Web site.

Each month, Zarko Gajic, About.com Guide to Delphi Programming, presents a Best of the Net Award to the top Delphi programming site on the Internet.

"The DelphiMag.com was found to be a comprehensive, credible, and informative site that simply could not be passed up," said Zarko Gajic, About.com Guide to Delphi Programming.

For more information about the award, visit http://delphi.about.com/compute/delphi/library/blbon2000.htm.

ICG Announces New Online Publications, Retitles Existing Sites

Elk Grove, CA — Informant Communications Group, Inc. (ICG), publisher of *Delphi Informant Magazine* and other technical magazines, has announced XML-Zine.com and SQLServerZine.com for launches in 2000.

Visitors to XML-Zine.com and SQLServerZine.com will be treated to tips and tricks, how-to articles, book reviews, product reviews, third-party product information, and industry news columns. These sites will feature file downloads and a membership-based forum, as well as easy navigation and a clean, easy-to-read design.

In addition to the announcement of the two new sites, ICG announced a name change for two of its most popular online magazines. DelphiMag.com and CBuilderMag.com will become DelphiZine.com and CBuilderZine.com, respectively, effective February 1, 2000.

"In an effort to provide uniformity to our online publications, ICG has replaced 'Mag' with 'Zine' in its growing number of Webzines," said Mitch Koulouris. "This change also enables each Webzine to stand out as an individual site — not necessarily attached to a print magazine."

ICG will continue to publish the award-winning *Delphi Informant Magazine* in its current form.

ICG is recognized as the leader in publishing quality technical magazines and Web sites. ICG produces *Microsoft Office & Visual Basic for Applications Developer*. The world's largest corporations, educational institutions, and government agencies read ICG's publications and Web sites, with readers in more than 60 countries. To serve the large Internet community, ICG has already launched OfficeVBA.com, DelphiZine.com, CBuilderZine.com, VJInformant.com, and ComputerBookstore.com, and anticipates several new IT professional sites soon. ICG also produces in-box product catalogs and hosts Microsoft Office/-VBA conferences worldwide.

For further information, please contact:

Joe Krack
Advertising Director
Phone: (916) 686-6610 x27
E-Mail: jkrack@informant.com

Subscriptions:
Customer Service Department
Phone: (916) 686-6610 x10
E-Mail:
circulation@informant.com

Author Inquiries:
Lori Ash
Acquisitions Editor
Phone: (916) 686-6610 x32
E-Mail: lash@informant.com

Press Releases/Product Announcements:
Chris Austria
Products Editor
Phone: (916) 686-6610 x16
E-Mail: caustria@informant.com

Inprise UK Sets Up Shop on the Web

London, England — Inprise UK, a subsidiary of Inprise Corp. announced the launch of the Inprise online shop for customers in the UK and Ireland. The new site will allow existing Inprise customers to purchase upgrades to any of the Borland development tools online and receive products with documentation directly through the post. Customers will be able to access the shop from http://shop.borland.co.uk.

By offering direct e-commerce capability, Inprise is providing customers with a one-stop shop for news, developer resources, and online purchasing through its Web resources.

Inprise's Customer Services Centre will handle the fulfillment of customer orders from its London-based warehouse and aims to dispatch orders within five working days. All online customers will have access to existing Inprise resources, including the Inprise customer services hotline and technical support.

table of CONTENTS

By Jani Järvinen

WAP! Delphi Does Wireless

Developing WML Applications for Mobile Phones

Delphi is well known for its ability to create HTML files. However, Delphi can also be used to create WML (Wireless Markup Language) files that are required by the new WAP (Wireless Application Protocol) services. In this article, you will learn how to create a real-time, order query system to be used with a WAP phone or a simulator. But we're getting ahead of ourselves a bit. Let's start at the beginning.

WAP is the protocol used by the newest WAP-enabled mobile phones. Many people think that WAP will bring the Internet to mobile phones. This isn't entirely true; after all, you can't surf the Web using a WAP phone. What WAP *will* do is allow you to retrieve and display special files developed for WAP-enabled phones. These files are coded using WML, which is analogous to HTML. Although rich formatting — such as nested tables, layers, colors, or different typefaces — can't be used in WML, WML does allow developers to specify simple text formatting, images, links, and text-entry fields. The reason for this simplicity is that the current mobile phones aren't capable of displaying rich formatting. The current WAP phones have tiny pixel displays (about 200 x 200) with about four shades of gray to represent colors. A new markup language was necessary for that kind of phone, because HTML files simply wouldn't fit on the screen.

The Protocol

WAP services are accessed by standard URLs. When the user enters the URL of a WAP service on his or her phone, the phone starts to communicate wirelessly with a WAP gateway. The gateway is a computer hosted by a mobile service operator, such as MCIWorldCom or Sprint. [At press time, a merger of MCIWorldCom and Sprint was pending regulatory and shareholder approval. — Ed.]

The gateway computer processes the mobile phone's request and redirects it to the normal Internet using HTTP. There, the service provider's Web server answers the request and returns WML data back to the WAP gateway. Then, the gateway sends the WML file back to the mobile phone, which eventually displays the file to the user.

Because HTTP is used to access WML files on the Internet, WAP technology doesn't require special hardware or software from the service provider. Ordinary Web servers, such as Apache and Microsoft Internet Information Server (IIS), can be configured to properly serve WML files. Furthermore, a single Web server can serve HTML and WML pages, just as the Web server can serve different file types, such as .gif, .jpeg, .zip, etc. However, a single (virtual) Web server should be dedicated to WAP, because this makes it easier for the user to remember the URLs.

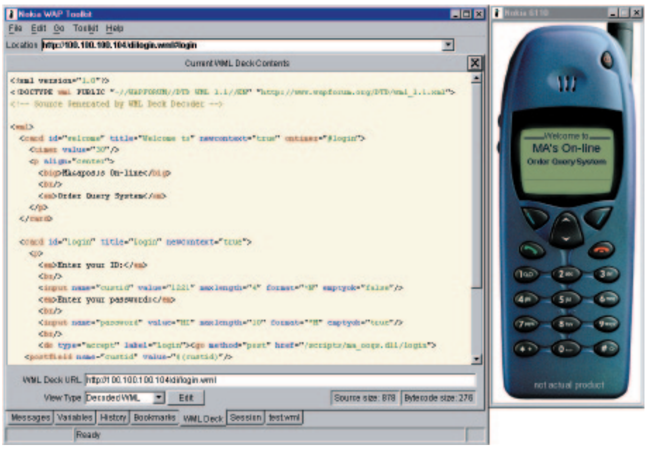
```
<?xml version="1.0 "?>
<!DOCTYPE wml PUBLIC "-// WAPFORUM// DTD WML 1.1// EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
<card id="welcome" title="Welcome to" newcontext="true"
      ontimer="#login">
  <timer value="30 "/>
  <p align="center">
  <big>MA's On-line</big><br/>
  <em>Order Query System</em>
  </p>
</card>
</wml>
```

Figure 1: An example of WML version 1.1 code.

The Language

WML is a language with XML syntax. This means that WML can be written using any text editor, such as Notepad. Also, WML resembles HTML, and many of the WML tags, such as <b>, <i>, <big>, and <a>, have almost the same meaning as in HTML. Figure 1 shows an example of WML code.

One way WML differs from HTML is that a file isn't called a "page." Rather, a single WML file is called a deck, and a single deck consists of one or more cards.


Nokia WAP Toolkit
File Edit Go Toolkit Help
Location http://100.100.100.104/di/login.wml#login
Current WML Deck Contents
WML Deck URL http://100.100.100.104/di/login.wml
View Type Decoded WML
Edit
Source size: 878
Bytecode size: 276
Messages Variables History Bookmarks WML Deck Session testwml
Ready
Welcome to
MA's On-line
Order Query System
not actual product

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<!-- Source Generated by WML Deck Decoder -->

<wml>
 <card id="welcome" title="Welcome to" newcontext="true" ontimer="#login">
  <timer value="30"/>
  <p align="center">
   <big>MA's On-line</big>
   <br/>
   <em>Order Query System</em>
  </p>
 </card>

 <card id="login" title="Login" newcontext="true">
  <p>
   <em>Enter your ID:</em>
   <br/>
   <input name="custid" value="1221" maxlength="4" format="*N" emptyok="false"/>
   <em>Enter your password:</em>
   <br/>
   <input name="password" value="HI" maxlength="10" format="*M" emptyok="true"/>
   <br/>
   <do type="accept" label="Login"><go method="post" href="/scripts/ma_ooqs.dll/login">
  <postfield name="custid" value="$(custid)"/>
```

Figure 2: The Nokia WAP Toolkit version 1.2. As Java-based software, it requires JRE 1.2.2.

Only one card is displayed on the phone's screen at a time, and a WML card can display generally only a few lines of text.

Current WAP phones can process fewer than 10 kilobytes of WML code at a time, so WML code files should be kept small. Also, because a very limited amount of text will fit on the screen, every WML card should be kept short. If the data on a WML card doesn't fit on the screen, the user is forced to scroll. The best current WAP services don't require extensive scrolling, but instead represent the information in small chunks. The user is then allowed to browse back and forth among different views of the data.

Browsing is allowed through ordinary hypertext links as well as something called tasks. Tasks are special commands written with the <do> WML tag. For example, the tags:

```
<do type='accept'> <prev/> </do>
```

would allow the user to browse backward in the WAP service.

To enable more interesting commands than simply browsing back and forth, WML can be extended with a scripting language called WMLScript. WMLScript is comparable to JavaScript in the HTML world, but, of course, it has a much more limited functionality. Still, WMLScript can be used to validate user input, evaluate simple equations, or set the WAP browser state.

The Toolkit

To help developers build WAP solutions, Nokia has developed a Java-based WAP Toolkit that can be used to create, test, and debug WAP services (see Figure 2).

The WAP Toolkit simulates a real WAP phone and the microbrowser software on it. The WAP Toolkit allows the loading of any WML file, from disk or the Internet. After loading the file, the WAP simulator processes it and displays the results on the screen. The developer can then use the mobile phone's buttons to enter text, select options, and navigate the WAP service.

Navigation can also be done using bookmarks, which the toolkit supports natively. Actually, bookmarks are only one simple feature to make the developer's life easier. For example, the toolkit allows real-time monitoring of variables created by the WML and WMLScript code. This is comparable to Delphi's "watches" support, although the toolkit provides much more limited functionality. Nonetheless, the variable view allows the developer to debug WML code more easily.

As the toolkit allows the developer to load any WML file using standard HTTP, the simulator is a great way to test interactive systems running on a Web server. As described previously, a normal Web server can easily serve WML and WMLScript pages. Of course, these pages can also be created dynamically.

Delphi 5 provides good support for building Web solutions with its integrated WebBroker technology. Normally, the WebBroker technology is used to create dynamic HTML solutions that run on many popular Web servers. However, given its flexibility, the WebBroker technology can also be used to create WML on the fly.

The Delphi Solution

The example program, which is available for download, is an ISAPI DLL, which runs on IIS 3.0 or later. The detailed workings of ISAPI DLLs are beyond the scope of this article, but you can find detailed information about ISAPI on the Internet. One of the best sources is Microsoft Developer Network (MSDN) at http://msdn.microsoft.com. For information on debugging ISAPI DLLs, see the sidebar "Debugging ISAPI DLLs" (on page 9).

By default, ISAPI DLLs return normal HTML code, which can be created manually, or by using the PageProducer components that ship with Delphi. Of course, the biggest advantage is that Delphi allows easy connection to databases, which can then be queried to retrieve the data inserted into the HTML code. Our example program does just this; it queries a database, creates WML to contain that information, and sends the results to the user. For simplicity, the example program uses the DBDEMOS database alias that ships with Delphi.

The DBDEMOS database allows access to a fictive customer and order database of a diving equipment reseller, Marine Adventures. The example program allows the WAP mobile phone user to view the orders of a given customer in real time. Given this, the example program is named "Marine Adventures On-line Order Query System," or MAOOQS for short (available for download; see end of article for details).

Logging in to Marine Adventures

When you want to use MAOOQS, enter `http://myserver/login.wml` into your WAP phone (or the WAP Toolkit in this case). The WML browser briefly displays a "Welcome to" screen, then proceeds to a Login screen. The WML code for the "Welcome to" and Login cards is shown in Figure 3.

The Login screen contains two input fields, one for the customer ID and one for the password. These fields are created using the WML <input> tag, and the `name` attribute specifies the variable name to which the data entered by the user gets assigned.

The <do> tag defines a <go> action, which points to the example program's ISAPI DLL (by default on the path /scripts/ma_ooqs.dll). The <go> action uses the standard HTTP `post` command to send the data entered by the user to the DLL. Note that the <postfield> tags automatically instruct the browser to encode the contents of the `$(custid)` and `$(password)` variables for transfer via HTTP.

After the user clicks the Login command, the phone contacts the example DLL. This causes the DLL to execute the login action listed in Figure 4. First, the code extracts the `custid` and `password` fields from the *Request.ContentFields* property. After the data entered

```
<card id="welcome" title="Welcome to" newcontext="true"
      ontimer="#login">
  <timer value="30 "/>
  <p align="center">
  <big>MA's On-line</big><br/>
  <em>Order Query System</em>
  </p>
</card>

<card id="login" title="Login" newcontext="true">
  <p>
  <em>Enter your ID:</em><br/>
  <input name="custid" value="1221" maxlength="4"
         format="*N" emptyok="false"/>
  <em>Enter your password:</em><br/>
  <input name="password" value="HI" maxlength="10 "
         format="*M" emptyok="true"/>
  <br/>
  <do type="accept" label="Login">
    <go method="post" href="/scripts/ma_ooqs.dll/login">
      <postfield name="custid" value="$(custid)"/>
      <postfield name="password" value="$(password)"/>
    </go>
  </do>
  </p>
</card>
```

Figure 3: The WML code for the "Welcome to" and Login cards in the login.wml file.

```
function TMAWebModule.IsValidCustID(
  CustID, Password: string): Boolean;
begin
  DebugMessage('IsValidCustID:'#13'CustID="' + CustID +
             #13'Password="' + Password +' "');
  Result := False;
  with Customer do begin
    try
      Open;
      if Locate('custno',CustID,[]) then
        if (LowerCase(CustomerState.AsString) =
            LowerCase(Password)) then
          Result := True;
    finally
      Close;
    end;
  end;
end;

procedure TMAWebModule.MAWebModuleLoginAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  CustID, Password : string;
begin
  CustID := Request.ContentFields.Values['custid'];
  Password := Request.ContentFields.Values['password'];
  { Check validity of custid/password. }
  with Response do begin
    if IsValidCustID(CustID,Password) then
      begin
        DebugMessage('CustID/Password is valid.');
        MAWebModuleMainMenuAction(Sender, Request,
                                  Response, Handled);
      end
    else
      begin
        DebugMessage('Invalid CustID/Password pair.');
        ContentType := MIMETypeWML;
        Content := InvalidIDPageWML;
      end;
  end;
  Handled := True;
end;
```

Figure 4: The code listing for the Login action and the *IsValidCustID* function.

by the user is sitting comfortably in string variables, the ID and password are validated by doing a database lookup. The lookup function *IsValidCustID* can also be seen in Figure 4.

Because the DBDEMOS database doesn't contain actual user ID and password pairs, the example program uses the CustNo field from the Customer table as the ID, and the State field as the password. For testing purposes, you can use the following ID/password pairs: "1221" and "HI"; "1560" and "FL"; and "1680" and "GA."

Returning WML to the Browser

Normally, WebBroker applications return simple HTML code to the browser by manipulating the *Response* object in the *OnAction* event handler. The *TWebResponse* class represented by the *Response* parameter contains a *Content* property, which accepts the HTML code created by the event handler.

When the user's browser accepts data, it processes it according to its MIME type (Multipurpose Internet Mail Extensions). By default, the MIME type of the *TWebResponse* class is "text/html," which specifies that the data in the *Content* property is, in fact, HTML code.

However, WAP applications require that the MIME type of the WML (version 1.1) data is "text/vnd.wap.wml." For this reason, the *ContentType* property of the *Response* object needs to be changed. Note that the *ContentType* property can be changed to any valid MIME type. For example, changing the property to "image/gif" allows the action to return binary .gif image data.

After the correct MIME type has been specified, the action event handler simply sets the *Content* property to contain valid WML code. Because every WML page generated by the application requires a common header, the example program defines a constant named WMLHeader to contain this information.

Storing Data Back to the Database

After the user's login has been validated, the user is redirected to the Marine Adventures main menu. This menu allows the user to choose to view customer information or order status. If the user chooses the customer information hyperlink, the execution branches to the *CustInfoAction* method of the *MAWebModule* Web module (see Figure 5).

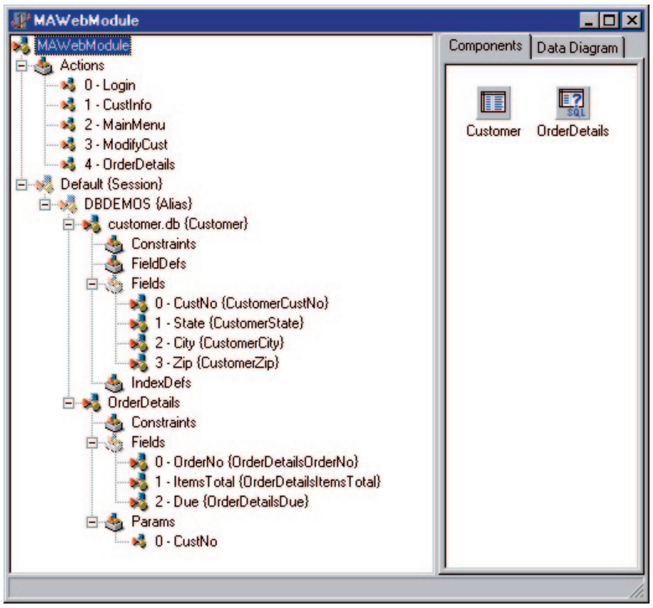


Figure 5: The *MAWebModule* in the Delphi IDE.

First, the code retrieves the *CustNo* parameter that was sent along with the POST HTTP command. After locating the customer specified by the parameter in the Customer table, the code constructs WML code to contain three input fields. Initially, these contain the customer's ZIP code, city, and state (remember, the state is the password).

The user is allowed to modify these fields, and by selecting the "Modify Info" command, the user can store the new values back to the database. When the user selects this command, the execution goes to the *ModifyCustAction* method (see Figure 6).

The code extracts the new customer information from the *Request.ContentFields* property, then locates the correct customer record in the Customer table. Next, the code puts the table into *dsEdit* state by calling the *Edit* method, sets the fields, and finally calls the *Post* method.

As you can see, there's nothing special about storing data back to a database in Web applications. The most difficult thing is to code the WML or HTML files so they correctly transfer the information entered by the user to the Web application.

```
procedure TMAWebModule.MAWebModuleModifyCustAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
  var CustID, State, City, Zip : string;
begin
  CustID := Request.ContentFields.Values['custid'];
  State := Request.ContentFields.Values['State'];
  City := Request.ContentFields.Values['City'];
  Zip := Request.ContentFields.Values['Zip'];
  with Response do begin
    ContentType := MIMETypeWML;
    try
      if LocateCustomer(CustID) then
        begin
          DebugMessage('Saving customer data:'#13 +
                       'CustID="' + CustID + '"'#13 +
                       'State="' + State + '"'#13 +
                       'City="' + City + '"'#13 +
                       'Zip="' + Zip + '"'#13);
          Customer.Edit;
          CustomerState.AsString := State;
          CustomerCity.AsString := City;
          CustomerZip.AsString := Zip;
          Customer.Post;
          Content := WMLHeader +
            '<card id="modifyok" title="Info ' +
            'Modified" newcontext="true">' + CRLF +
            '  <onevent type="ontimer">' + CRLF +
            '    <go method="post" ' +
            'href="/scripts/ma_ooqs.dll/mainmenu">' +
            CRLF + '      <postfield name="custid" ' +
            'value="' + HTTPEncode(CustID) + '"/>' + CRLF +
            '    </go>' + CRLF + '  </onevent>' + CRLF +
            '  <timer value="30"/>' + CRLF + '  <p>' +
            CRLF + '  <b>Customer info succesfully '+
            'modified.</b><br/>' + CRLF + '  </p>' + CRLF +
            '</card>' + CRLF + CRLF + '</wml>' + CRLF;
        end
      else
        Content := InvalidIDPageWML;
    finally
      Customer.Close;
    end;
  end;
  Handled := True;
end;
```

Figure 6: Storing data entered by the user back to the Customer table.

Debugging ISAPI DLLs

When debugging ISAPI DLLs, it's often helpful to display simple messages on the screen. Of course, IIS is a service, so normal modal forms can't be displayed. Still, you can use the Windows **MessageBox** API call to display simple text on the screen. All you have to do is include the MB_TOPMOST and MB_SERVICE_NOTIFICATION flags in the *uType* parameter, as shown in the following code:

```
procedure DebugMessage(S: string);
begin
  MessageBox(0 ,PChar(S),'My ISAPI DLL',
    mb_OK + mb_Topmost + mb_Service_Notification);
end;
```

— Jani Järvinen

Testing Marine Adventures

To test the example application, you need to have a Web server capable of running ISAPI applications. The example application has been tested with Microsoft IIS 4.0, but it should also work with other compatible products or different versions of IIS.

Besides the Web server, you need to download and install the Nokia WAP Toolkit version 1.2 (at press time, Nokia WAP Toolkit 1.3 beta is available for download). This can be downloaded free from Nokia's WAP site at http://www.forum.nokia.com/. The toolkit doesn't need to be installed on the same computer as the Web server; all that's needed is an IP network connection to the Web server.

When the WAP Toolkit is fired up, it displays a welcome project (refer to Figure 2). The Toolkit can simulate two mobile phone models. For the purposes of MAOOQS, select phone model 6110 using the **Toolkit | Preferences** menu command.

To navigate to the Marine Adventures Login screen, choose the **Load Location** command from the **Go** menu. Type the URL of the login.wml file: `http://localhost/login.wml`. Note that before doing this, the login.wml file needs to be copied to the root of the Web server's publish directory. It should go without saying that the Web server must also be running to successfully run MAOOQS.

Figure 7 displays the corresponding screens involved in testing our sample Marine Adventures application. After the login.wml file has been loaded, you will briefly see the "Welcome to" screen shown in Step 1. In a few seconds, the screen changes automatically to that shown in Step 2. To enter text into the input fields, click the blue backslash button on the top-left corner of the phone's keypad. When clicking the backslash button, the lower-left corner of the screen on the phone should read "Edit." If it instead says "Login," you'll need to click the up or down arrow button, because there's a time-out associated with the Edit command.

After successfully clicking the Edit command, the screen changes to display the integrated text field editor shown in Step 3. The customer ID field accepts only numbers, so clicking on any of the numbered buttons adds the corresponding number to the field. If you make a typing mistake, the slash button on the right can be used to clear the last character.

The password field accepts characters and numbers, so you have to click the numbered buttons repeatedly to enter different characters. For example, to enter the character "B," you need to press the 1

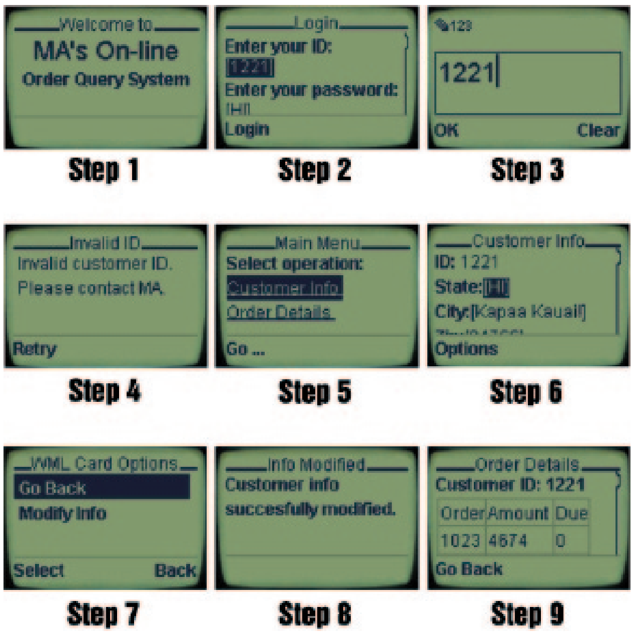


Figure 7: Screens of the Nokia 6110 simulator. The steps indicate the order of appearance in MAOOQS.

button two times in rapid succession. If you've used a mobile phone to type text, you'll recognize this style of typing.

Requesting Data from the DLL

After the customer ID and password have been successfully entered, you should again find yourself at the screen shown in Step 2. Wait briefly for the "Login" command to become active, then click the backslash button to select the command. This will proceed with the login.

Selecting the Login command causes the WAP simulator to connect to the MA_OOQS ISAPI application on the Web server's /scripts/ directory. Of course, this requires that the ISAPI DLL has been placed on the Web server's /scripts/ directory and that the Web server has enough permissions to properly execute the DLL.

If everything goes smoothly, the WAP simulator receives proper WML data from the DLL. If an incorrect customer ID and/or password were entered, the Invalid ID screen appears as shown in Step 4. Otherwise, the screen displays the Main Menu as shown in Step 5.

The arrow keys on the phone allow you to select either one of the menu commands. By clicking the backslash button, you can go to the Customer Info screen (Step 6), or the Order Details screen (step 9). Pressing the backslash button again allows you to toggle among these three pages. If you change the customer information with the Modify Info command shown on the screen in step 7, you will briefly see the Info Modified screen shown in Step 8.

It's also worthwhile to note that WAP services usually don't have a "log out" command, because such a command isn't necessary. To go to a different WAP service, simply enter its URL, just as you would with your favorite Web browser.

Conclusion

Currently, WAP is the hottest thing happening in the area of mobile phones and wireless Internet. Although it requires studying new technology, only the basic understanding of the technology is required to implement attractive WAP solutions.

Furthermore, knowledge of existing HTML and HTTP technologies is of great help, because WML and WMLScript aren't actually that different from HTML and JavaScript. Again, Delphi provides great tools for building WAP solutions, because the WebBroker technology present in Delphi can be extended to support WAP.

This article shows how WAP solutions can be built with Delphi. Although all the details can't be discussed here, you should now understand how such services work. With some hacking, you should be able to build your own WAP solutions to support the constantly growing needs of mobile users. ∆

If you would like more information on this burgeoning technology, visit the WAP forum (http://www.wapforum.org), Nokia Corp. (http://www.nokia.com), Ericsson (http://www.ericsson.com), or Siemens (http://www.siemens.com).

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\APR\DI200004JJ.

Jani Järvinen works as a technical support person for Borland development tools. He also develops custom Internet software.

By Cary Jensen, Ph.D.

Interfaces Revisited

Part II: Interface References versus Object References

Last month, I began a two-part series on interfaces. As you recall, an interface is a declaration of methods and properties that can be implemented by a class.

The primary advantage of interfaces is that they permit assignment compatibility between two or more objects that implement a common interface. In other words, interfaces permit two objects to be treated polymorphically with respect to the methods and properties of the interface, even though those objects do not inherit the methods and properties from a common ancestor. In this respect, interfaces provide crucial support for polymorphism in languages, such as Object Pascal, that don't support multiple inheritance. Furthermore, from an object-oriented design viewpoint, interfaces provide an elegant way to define an object's behavior independently of the object's implementation.

In Part I, I described the basic rules of interface declaration and class implementation. In this installment, we'll look at using objects through interface references, and show how this usage differs from normal object usage. I'll also describe the additional support for interface implementation that was added to Delphi 4. This support, called "interface implementation by delegation," completes Delphi's interface model. This article concludes with a discussion of the two types of interface implementation by delegation, where I will argue that only one form of interface implementation by delegation is safe to use, while the other is inherently dangerous.

```
function TInterfacedObject.QueryInterface(
  const IID: TGUID; out Obj): HResult;
const
  E_NOINTERFACE = HResult($80 0 0 40 0 2);
begin
  if GetInterface(IID, Obj) then
    Result := 0
  else
    Result := E_NOINTERFACE;
end;

function TInterfacedObject._AddRef: Integer;
begin
  Result := InterlockedIncrement(FRefCount);
end;

function TInterfacedObject._Release: Integer;
begin
  Result := InterlockedDecrement(FRefCount);
  if Result = 0 then
    Destroy;
end;
```

Figure 1: *IUnknown* methods as implemented by the *TInterfacedObject* class.

Interface References versus Object References

There are significant differences between using interface references and using object references. The greatest of these is related to life-cycle management. Specifically, objects accessed through an interface are reference counted. When you assign an object to an interface reference, internally its *_AddRef* method is called. Similarly, when the interface reference is released, the object's *_Release* method is invoked.

All objects that implement interfaces must implement *_AddRef* and *_Release*. This is because all interfaces ultimately descend from *IUnknown*, and (as discussed in last month's article), any object that implements an interface must also implement all methods declared in that interface's ancestor interfaces. In the implementation of *_AddRef*, the object should increment an internal reference counter. From *_Release*, this reference counter is decremented. Furthermore, when *_Release* decrements the counter to zero, the object is explicitly released by *_Release*. An example of this type of implementation is shown in the *IUnknown* methods as implemented by the *TInterfacedObject* class shown in Figure 1.

Internally, the compiler generates the necessary call to *QueryInterface* in response to the assignment of an interface-implementing object to an interface reference. As shown in Figure 1, *TInterfacedObject.QueryInterface* invokes *GetInterface*, which is a method of the *TObject* class. *GetInterface* invokes *_AddRef* on the object whose interface is queried.

```
type
  IShowMessage = interface(IUnknown)
    ['{ F8AA90 A1-EA2D-11D0 -82A8-444553540 0 0 0  }']
    function ShowMessage:Boolean;
    function GetMessageText: string;
    procedure SetMessageText(Value: string);
    property MessageText: string
      read getMessageText write setMessageText;
  end;
  TYesDefault = class(TInterfacedObject, IShowMessage)
    FMessageText: string;
    function ShowMessage: Boolean;
    function GetMessageText: string;
    procedure SetMessageText(value: string);
  public
    destructor Destroy; override;
  end;
  TNoDefault = class(TInterfacedObject, IShowMessage)
    FMessageText: string;
    function ShowMessage: Boolean;
    function GetMessageText: string;
    procedure SetMessageText(value: string);
  public
    destructor Destroy; override;
  end;
```

Figure 2: Declaring an interface and two classes that implement it.

The compiler also generates calls to *_Release*. There are three situations in which the compiler generates the *_Release* invocation. These are:

- The interface reference is assigned a value of **nil**.
- The interface reference is re-assigned a different interface-implementing object.
- The interface reference goes out of scope.

As you can see in Figure 1, when *_Release* is invoked and the reference count drops to zero, the interface object's destructor is invoked automatically.

This behavior, which only occurs when using interface references, is substantially different from the behavior observed when object references are used. Using object references, the life cycle of objects is managed in one of two ways: Either you rely on the *TComponent* garbage collection, in which the *Owner* of an object (the one passed to the *TComponent.Create* method) destroys all existing owned objects (i.e. those objects referenced in the owner's *Components* array property) as part of its own destruction, or you explicitly call *Free* (or *Release* for *TForm* instances). If the object reference is not a *TComponent* descendant, then you must explicitly call *Free* to release the object.

These differences in life-cycle management result in substantially different-looking code, depending on whether you're using interface references or object references. These differences are found in the sample application named DEMOINT (available for download; see end of article for details). You'll want to download this file and inspect the entire unit for the main form. For brevity, I'm only going to focus on parts of this file at any given time.

First, let's consider the declaration of one interface and two classes that each implement the interface. For clarity, I have kept this interface and its implementing class declarations very simple, as shown in Figure 2.

The first thing you'll notice is that the *IShowMessage* interface is associated with a large number. This number is referred to as the interface's globally unique identifier, or GUID for short (pronounced "goo-id"). This is a 128-bit number generated through a call to a Windows API

```
function TYesDefault.GetMessageText: string;
begin
  Result := FMessageText;
end;

procedure TYesDefault.SetMessageText(value: string);
begin
  FMessageText := Value;
end;

function TYesDefault.ShowMessage;
begin
  if MessageBox(Application.Handle, 'Continue?',
                PChar(FMessageText),
                MB_OKCANCEL+MB_DEFBUTTON1) <> IDOK then
    Result := False
  else
    Result := True;
end;

destructor TYesDefault.Destroy;
begin
  Dialogs.ShowMessage('YesDefault: Goodbye');
  inherited;
end;

function TNoDefault.GetMessageText: string;
begin
  Result := FMessageText;
end;

procedure TNoDefault.SetMessageText(value: string);
begin
  FMessageText := Value;
end;

function TNoDefault.ShowMessage;
begin
  if MessageBox(Application.Handle, 'Continue?',
                PChar(FMessageText),
                MB_OKCANCEL+MB_DEFBUTTON2) <> IDOK then
    Result := False
  else
    Result := True;
end;

destructor TNoDefault.Destroy;
begin
  Dialogs.ShowMessage('NoDefault: Goodbye');
  inherited;
end;
```

Figure 3: Implementing *TYesDefault* and *TNoDefault*.

function, which assures that this number will be absolutely unique, even across different computers. The purpose of the GUID is to uniquely identify this interface when it is registered for use with COM (Component Object Model) in the Windows registry.

Object Pascal doesn't require you to generate and register a GUID for interfaces you create. However, in case you decide to use the interface as a COM interface sometime in the future, it doesn't hurt to assign a GUID. And Delphi's editor makes generating the GUID very easy. Simply press Ctrl Shift G to insert a GUID into your source code.

The remainder of this interface is pretty simple. It declares a single property, named *MessageText*, and three methods. Two of these three methods are accessor methods for the property.

As you learned in Part I of this series, interfaces do not implement methods. This is the responsibility of any class that implements the interface. In Figure 2, both the *TYesDefault* and *TNoDefault* classes

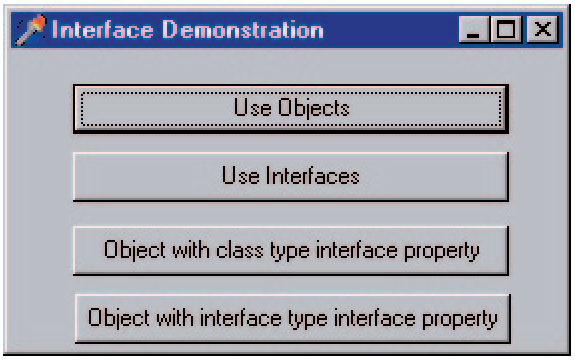

Interface Demonstration
Use Objects
Use Interfaces
Object with class type interface property
Object with interface type interface property

Figure 4: The main form of the DEMOINT project.

implement the *IShowMessage* interface. Because neither *TYesDefault* nor *TNoDefault* inherit the methods of the *IShowMessage* interface, both classes must explicitly declare and implement these methods. Notice, however, that the *MessageText* property is not declared in either *TYesDefault* or *TNoDefault*. As mentioned in Part I, classes that implement an interface are not required to explicitly declare the interface's properties. Doing so is entirely optional, and I opted not to in this case. However, because these classes must implement the accessor methods *GetMessageText* and *SetMessageText*, and these methods require the storage of the text of the message, it was necessary to declare a member field, named *FMessageText* in this case, to hold the value of the message in these class instances.

Although these two classes look nearly identical, they are different. The difference in this case is found in their implementations of the *ShowMessage* method. Figure 3 shows the implementations for both of these classes. I've also overridden the destructors of both of these classes, so we can see the results of life-cycle management, and compare how this differs between interface references and object references.

As mentioned earlier, the only real difference between the implementation of these two classes is found in the *ShowMessage* method. When *TYesDefault.ShowMessage* is invoked, a dialog box with a default OK button is displayed. By comparison, when *TNoDefault.ShowMessage* is invoked, the Cancel button is the default button. This single difference permits us to tell which implementation of the interface is being invoked by noticing which button on the displayed dialog box is the default button.

Let's now turn our attention to the use of these two objects. The DEMOINT main form contains four buttons (see Figure 4). Begin by considering the first two, labeled Use Objects and Use Interfaces, respectively. Figure 5 is the *OnClick* event handler associated with the button labeled Use Objects.

Here we see the traditional way to use an instance of a class. The class constructor is called, its methods are invoked, and then it is freed. Notice that we use the accessor methods to set the text of the dialog box's message. We could not use the *MessageText* property in this case, because that property is not a property of either the *TYesDefault* or *TNoDefault* class.

Let's compare this to the code associated with the button labeled Use Interfaces. Figure 6 shows the code attached to this button's *OnClick* event handler.

Here we're using a variable of type *IShowMessage* — an interface reference. Furthermore, we're able to use the *MessageText* property to set the text of the displayed message, because the *IShowMessage* interface has this property. Other than these two differences, there are two other important differences between this code segment and the preceding one. First, a single variable, of type *IShowMessage*, was used to reference both object instances. In other words, both the *TYesDefault* and the *TNoDefault* instances are assignment compatible with the *IShowMessage* reference.

The second important difference is that these objects were not explicitly freed. Instead, their life cycle was managed by the interface reference. You'll see this if you download and run this project. When you click the Use Interfaces button, you'll first see a YesDefault dialog box. Once you click one of the buttons on this dialog box, you'll see a message displayed by the destructor of the YesDefault instance. This destructor is invoked when the *TNoDefault* instance is assigned to the *IShowMessage* variable. In other words, when the interface reference is directed to another interface-implementing object, the first reference was released, resulting in its implicit destruction. After the destructor's message, an instance of the NoDefault dialog box is displayed. Again, after clicking a button on this dialog box, you'll see the message displayed by the destructor of the NoDefault instance. In this case, this occurs because the *IShowMessage* variable went out of scope.

Interface Implementation by Delegation

With Delphi 4, an important new mechanism for implementing interfaces was introduced. Rather than having to explicitly declare and implement each method of the interface, you could declare a property of either a class type or an interface type and delegate the implementation of the interface to that property. When the implementing object is assigned to the interface reference, it is the object assigned to the interface property that is provided to the interface reference.

At first this addition to interface support might sound unnecessary, but it's actually very useful. An example will demonstrate why. Imagine that you have 20 different objects that must implement a particular interface. Assuming these objects do not inherit the methods of

```
procedure TForm1.Button1Click(Sender: TObject);
var
  YesDefault1: TYesDefault;
  NoDefault1: TNoDefault;
begin
  YesDefault1 := TYesDefault.Create;
  NoDefault1  := TNoDefault.Create;
  YesDefault1.SetMessageText('This is a TYesDefault');
  YesDefault1.ShowMessage;
  NoDefault1.SetMessageText('This is a TNoDefault');
  NoDefault1.ShowMessage;
  YesDefault1.Free;
  NoDefault1.Free;
end;
```

Figure 5: The *OnClick* event handler for the **Use Objects** button.

```
procedure TForm1.Button2Click(Sender: TObject);
var
  IntVar: IShowMessage;
begin
  IntVar := TYesDefault.Create;
  IntVar.MessageText := 'This is a TYesDefault';
  IntVar.ShowMessage;
  IntVar := TNoDefault.Create;
  IntVar.MessageText := 'This is a TNoDefault';
  IntVar.ShowMessage;
end;
```

Figure 6: The *OnClick* event handler for the **Use Interfaces** button.

the interface, in Delphi 3 it was necessary for each of the implementing classes to declare and implement the methods of the interface. Assuming that each of the 20 classes implemented the interfaces in exactly the same way, you have 20 different copies of the same code. If you decide that the common implementation must be changed, you have to change your code in 20 different classes.

Interface implementation by delegation solves this problem. Instead of declaring and implementing the interface in each of the 20 classes, you can create a single class that implements the interface. Then, in each of the 20 classes that must implement this same interface, you declare a property that can hold a reference to the single class you initially created. In other words, all 20 classes can share the implementation provided for by your single class. If you later need to change how the interface is implemented, you simply return to your single class and change its implementation. Because all 20 classes delegate the implementation to this single class, their implementation is automatically updated.

```
type
  TClassProperty = class(TComponent, IShowMessage)
  private
    FMyMessage: TYesDefault;
  public
    property MyMessage: TYesDefault
      read FMyMessage write FMyMessage
      implements IShowMessage;
  end;
  TInterfaceProperty = class(TComponent, IShowMessage)
  private
    FMyMessage: IShowMessage;
  public
    property MyMessage: IShowMessage
      read FMyMessage write FMyMessage
      implements IShowMessage;
  end;
```

Figure 7: The *TClassProperty* and *TInterfaceProperty* classes.

```
procedure TForm1.Button3Click(Sender: TObject);
var
  IntVar: IShowMessage;
  ClassProperty1: TClassProperty;
begin
  ClassProperty1 := TClassProperty.Create(Self);
  ClassProperty1.MyMessage := TYesDefault.Create;
  IntVar := ClassProperty1;
  IntVar.MessageText := 'Message Text';
  IntVar.ShowMessage;
  ShowMessage('About to free the interface');
  IntVar := nil;
  ShowMessage(
    'Interface freed. About to release the object');
  ClassProperty1.Free;
  ShowMessage('Object freed');
end;

procedure TForm1.Button4Click(Sender: TObject);
var
  IntVar: IShowMessage;
  InterfaceProperty1: TInterfaceProperty;
begin
  InterfaceProperty1 := TInterfaceProperty.Create(Self);
  InterfaceProperty1.MyMessage := TNoDefault.Create;
  IntVar := InterfaceProperty1;
  IntVar.MessageText := 'Message Text';
  IntVar.ShowMessage;
  ShowMessage('About to free the interface');
  IntVar := nil;
  ShowMessage(
    'Interface freed. About to release the object');
  InterfaceProperty1.Free;
  ShowMessage('Object freed');
end;
```

Figure 8: The *OnClick* event handlers for the class-type interface and interface-type interface buttons.

As already mentioned, there are two types of properties you can use to implement an interface by delegation: class-type properties, and interface-type properties. Furthermore, when you declare a property to implement an interface by delegation, the property syntax must include the **implements** directive, followed by the name of the interface the property is implementing.

The DEMOINT project contains two classes that implement the *IShowMessage* interface by delegation. The first class, *TClassProperty*, implements the interface using a class-type property. The second, named *TInterfaceProperty*, implements the *IShowMessage* interface using an interface property. The code in Figure 7 shows the declaration of these two classes.

Notice that these class declarations include the name of the *IShowMessage* interface following the ancestor class, just as if each class was going to declare and implement the methods of the *IShowMessage* interface. However, the implementation of this interface is delegated to a property, named *MyMessage*, in both classes. Notice further that the syntax of the property declarations includes the **implements** directive followed by the *IShowMessage* interface name. This is required so the compiler can tell that you want the object assigned to this property to provide the implementation of the *IShowMessage* interface.

Both of these class declarations also include a private member field. This field is used to hold a reference to the object assigned to the associated property using direct access. In the case of the *TClassProperty.MyMessage* property, this member field is of a class type. By comparison, the *TInterfaceProperty.MyMessage* uses a member field of type *IShowMessage*.

The use of these two classes is demonstrated by the buttons on the DEMOINT main form labeled **Object with class type interface property** and **Object with interface type interface property**, respectively. Figure 8 shows the code associated with the *OnClick* event handlers of these two buttons.

For both of these event handlers, the first line creates an instance of the object that implements the interface through delegation, and the second line assigns an object to the interface property. In reality, the object that implements the interface would likely be assigned to the interface property in the constructor of the class that implements through delegation. For example, it would have been just as valid, and arguably more appropriate, to assign the *TYesDefault* instance to the *MyMessage* property of the *TClassProperty* instance from within the *TClassProperty* constructor.

The remaining statements in these event handlers are also similar. An object is assigned to the interface variable in the third statement, after which this reference is used to assign the message text and show the message. If you run this example, you'll see that clicking the **Object with class type interface property** button displays the implementation of the *TYesDefault* class, and clicking the **Object with interface type interface property** button displays the implementation provided for by the *TNoDefault* class.

The remaining five statements in each of these event handlers have been added to document the life cycles of the objects assigned to the interface-implementing properties. This is described in the following section.

Comments on Interface Implementation by Delegation

At first glance, there appears to be little difference between these two types of interface delegation. Both make use of an external object to provide for the implementation of the interface. However, there is a very big difference, one that leads me to strongly advise against delegating interface implementation to a class-type property.

The problem can be seen clearly if you compare the sequence of dialog boxes displayed by clicking on the two interface delegation buttons on the DEMOINT main form. Specifically, if you click **Object with class type interface property**, you'll see the following dialog boxes, in this order: the *TYesDefault ShowMessage* dialog box, the dialog box indicating that the interface is about to be freed, the *TYesDefault* destructor dialog box, the dialog box indicating the object is about to be freed, and finally the dialog box indicating that the object was freed.

Compare this sequence to the following, which occurs when you click on the button labeled **Object with interface type interface property**: the *TNoDefault ShowMessage* dialog box, followed by the dialog box indicating that the interface is about to be freed, then the dialog box indicating the object is about to be freed, followed by the *TNoDefault* destructor dialog box, and finally the dialog box indicating that the object was freed.

The difference is that freeing the interface variable caused the object assigned to the *TClassProperty.MyMessage* property to be released, but had no effect on the object assigned to the *TInterfaceProperty.MyMessage* property. The object assigned to *TInterfaceProperty.MyMessage* did not get released until the *TInterfaceProperty* instance was freed. It is this latter behavior that is correct.

The basic problem with the behavior when you use class-type properties is that you cannot reliably use the interface of an object that implements the interface using a class property, unless you first test to see if an object is still assigned to that property. Having previously assigned an object to the property is not sufficient, because an interface reference may have subsequently destroyed the object. In contrast, once an object has been assigned to an interface-type property that implements an interface, it is likely to still be there unless you've explicitly destroyed it from within your object.

The explanation for the difference in behaviors when using class-type versus interface-type properties is that the class that implements an interface using an interface-type property holds an internal reference to the object assigned to the property. In the *TInterfaceProperty* class, this reference is the *FMyMessage* member field. By comparison, unless you go to the extra trouble of adding an interface reference to the member field of a class-type property, the only reference counting will be provided by external interface references. Once this reference is released, the object assigned to the property is destroyed, rendering the implementing class unusable, from the perspective of the interface.

Interfaces and COM

If Delphi introduced interfaces for the purpose of supporting COM, you might be inclined to wonder why this series has been able to discuss interfaces with little discussion of COM. The answer is that interfaces are a valuable tool for treating objects polymorphically. In short, if you know that an object supports a given interface, you really know all you need to know about it in order to invoke the methods of that interface. This is important in COM because the objects you work with are often of an unknown origin.

COM is a binary standard, meaning that the objects you work with can be compiled using any compiler, not just Delphi's. In fact, when using COM, the objects you work with are normally compiled by something other than Delphi, such as Visual C++. But that's okay. As long as you use standard OLE types for your parameters, your Delphi applications will be able to communicate with these objects, invoking their behaviors and reading and writing their properties.

But interfaces are just one aspect of COM. There are many other issues, including how to provide support for a variety of threading models supported by COM servers, OLE types, and type libraries, among others. Consequently, I will not say more about COM in this article. For more information on COM, OLE automation, and ActiveX, refer to other articles, past and future, in *Delphi Informant Magazine*.

Interfaces and Delphi's Open Tools API

At the beginning of Part I of this series, I mentioned that one reason that interfaces deserved another look is Delphi's own increased reliance on them. Specifically, with each new version of Delphi, more and more of its own behavior is defined by interfaces. A good example of this is Delphi's Open Tools API. In Delphi 5, more than ever, creating your custom extensions to Delphi's IDE requires that you create classes that implement specific interfaces. For example, to add a custom key binding to Delphi's editor, you create and register a class that implements the *IOTAKeyboardBinding* interface (OTA stands for Open Tools API).

Conclusion

Interfaces provide you with the ability to treat objects polymorphically in the absence of inheritance. In other words, an interface defines the capabilities of an object without saying anything about how those capabilities are implemented. This is the essence of pluggable software, where one object asks for another object that implements a specific interface. How that interface is implemented is completely irrelevant. As long as the two objects understand the interface, they can work together. ∆

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\APR\DI200004CJ.

Cary Jensen is president of Jensen Data Systems, Inc., a Houston-based database development company. He is co-author of 17 books, including *Oracle JDeveloper* [Oracle Press, 1998], *JBuilder Essentials* [Osborne/McGraw-Hill, 1998], and *Delphi in Depth* [Osborne/McGraw-Hill, 1996]. He is a Contributing Editor of *Delphi Informant Magazine*, and an internationally respected trainer of Delphi and Java. For more information, visit http://www.jensendatasystems.com, or e-mail Cary at cjensen@compuserve.com.

By Ray Lischner

Sets to Strings, and Back

Employing Sets, Strings, and Run-time Type Information

A frequently asked question is: "How can I store a font style in the registry?" You have a number of choices; one approach is to convert the style to a string and store the string. You'll then need to convert the string back to a font style when loading the information from the registry. It's easy to hard-wire a couple of functions to do this for font styles, but it would be more useful to write general-purpose subroutines that work for any set. This article shows you how to take advantage of Delphi's Run-time Type Information (RTTI) to write functions that do just that: convert any set to a string and back again.

About RTTI

RTTI lies at the heart of Delphi's integrated development environment. The difference between a public property, method, or field and a published one is that published declarations have RTTI information that is available at run time. The Object Inspector uses RTTI for published properties to help it get and set property values. The form designer uses RTTI for published fields to define a field for each component you drop on a form.

RTTI also lets Delphi manage the lifetime of strings, dynamic arrays, interfaces, and variants. Even if you declare a dynamic array of strings in a record, for example, Delphi can navigate RTTI for the record, array, and string types to decide when it must initialize and finalize the strings and the dynamic array.

A set's RTTI contains a pointer to RTTI for the type that makes up the components of the set — usually an enumerated type, but it can also be an integer or character type. This article examines in depth RTTI for a set type. If you have the Professional or Enterprise Edition, you can read the source code in \Source\Vcl\TypInfo.pas; if you have the Standard Edition, read the interface in \Doc\TypInfo.int. Either file contains the information you need to make sense of RTTI.

A type's RTTI has two parts: type information and type data. The type information is stored in a *TTypeInfo* record, which contains a type kind and the type's name. The type kind is an enumeration of the various kinds of types in Delphi: *tkInteger*, *tkChar*, *tkEnumeration*, *tkSet*, and so on. The type kinds are self-explanatory. The built-in *TypeInfo* function returns a pointer to a type's *TTypeInfo* record.

TTypeData stores the type data in a variant record. Each type kind refers to different members of the record. For example, *tkSet* has a pointer to a *PTypeInfo* for the component type. Ordinal types have the *MinValue* and *MaxValue* members to store the range of values defined for the type. See the declaration in TypInfo.pas (or TypInfo.int) for the details of other types' data. If an ordinal type is declared as a subrange

Bit 7				Bit 3			Bit 0
unused	unused	unused	unused	*fsStrikeOut*	*fsUnderline*	*fsItalic*	*fsBold*

Figure 1: Bit representation of *TFontStyles*.

Bit 15							Bit 8	Bit 7							Bit 0
unused	unused	unused	unused	unused	10	9	8	7	unused	unused	unused	unused	unused	unused	unused
Byte 1								Byte 0							

Figure 2: Bit representation of an integer set.

```
const
  MaxSet = 255;  // Largest ordinal value in a Delphi set.
  BitsPerByte = 8;
type
  TSet = set of 0 ..MaxSet;

function SetToString(Info: PTypeInfo; const Value):
  string; overload;
begin
  Result := SetToString(Info, Value, ',');
end;

function SetToString(Info: PTypeInfo; const Value;
  const Separator: string): string; overload;
begin
  Result := SetToString(Info, Value, Separator, '[', ']');
end;

function SetToString(Info: PTypeInfo; const Value;
  const Separator, Prefix, Suffix: string):
  string; overload;
var
  CompInfo: PTypeInfo;
  CompData: PTypeData;
  SetValue: TSet absolute Value;
  Element: 0 ..MaxSet;
begin
  CompInfo := GetTypeData(Info)^.CompType^;
  CompData := GetTypeData(CompInfo);
  Result := '';
  for Element := CompData.MinValue to CompData.MaxValue do
    begin
      if Element in SetValue then
        if Result = '' then
          Result := Prefix + GetEnumName(CompInfo, Element)
        else
          Result := Result + Separator +
                    GetEnumName(CompInfo, Element);
    end;
  if Result = '' then
    Result := Prefix + Suffix
  else
    Result := Result + Suffix;
end;
```

Figure 3: First, simple implementation of *SetToString*.

```
// Convert an ordinal value to a string. The ordinal value
// can be an integer, enumerated value, or a character.
function OrdToString(Info: PTypeInfo; Value: Integer):
  string;
resourcestring
  sCvtError =
    'OrdToString: type kind must be ordinal, not %s';
const
  AsciiChars = [32..127];  // Printable ASCII characters.
begin
  case Info.Kind of
    tkInteger:
      Result := IntToStr(Value);
    tkChar, tkWChar:
      if Value in AsciiChars then
        Result := '''' + Chr(Value) + ''''
      else
        Result := Format('#%d', [Value]);
    tkEnumeration:
      Result := GetEnumName(Info, Value);
  else
    raise EConvertError.CreateFmt(sCvtError,
      [GetEnumName(TypeInfo(TTypeKind), Ord(Info.Kind))]);
  end;
end;
```

Figure 4: *OrdToString* converts any ordinal value to a string.

of another type, the type data's *BaseType* member points to the *PTypeInfo* for the base enumerated type.

Note that the pointers in *TypeInfo* all use two levels of indirection. For example, a *TTypeData* member might have the type *PPTypeInfo* instead of *PTypeInfo* (that is, pointer to pointer to *TTypeInfo* instead of a simple pointer to *TTypeInfo*). If you use packages, the type information for different types can reside in different packages. The extra level of pointers makes it easy for Windows to load the package DLL at any starting address and fix up RTTI pointers.

Sets in Delphi

Delphi stores a set value as a bit mask. A set can contain up to 256 elements, which requires a bitmask with 256 bits. At eight bits per byte, that means the largest possible set occupies 32 bytes. Delphi uses only as many bytes as it needs, so a set of up to eight elements usually fits in a single byte. For example, *TFontStyles* is a set with up to four members, occupying bits 0 through 3. Delphi ignores the most significant four bits in the byte because they aren't needed. Figure 1 illustrates the arrangement of bits in a byte.

I wrote "usually" in the previous paragraph because sets of integers or characters aren't always aligned on byte boundaries. Delphi stores sets so that a member's bit position is always the same for a given ordinal value. In other words, ordinal value 10 always sits at the third bit from the right in a byte. If the set type is, say, "set of 7..10", Delphi stores that set in two bytes, even though it fits in only four bits. The last bit of the first byte stores member 7, and the first three bits of the next byte store members 8, 9, and 10, as shown in Figure 2. This representation of sets results in compact storage and code, and makes it easy to assign set values and test set membership.

This representation for sets also makes it possible to write general-purpose subroutines that can handle any set. The subroutines need RTTI for the set type. The set type's component type specifies the *MinValue* and *MaxValue* for the ordinal type, and those values dictate the bitwise representation for the set. Delphi restricts the members of a set to ordinal values in the range 0..255, which means you can treat any set as a set of integers where the integers are a subrange of 0..255. For example, *TFontStyles* is equivalent to a set of 0..3, where 0 is the ordinal value of *fsBold*, 1 is *fsItalic*, and so on. The functions described in this article rely on this trick: Every set is treated as a set of integers, using RTTI to learn the true enumerated literal for each set member.

SetToString Function

The first — and easier — task is to convert a set to a string. The *SetToString* function requires the *PTypeInfo* pointer for the set type and the set value. To write a subroutine that takes any set as an argument, it must use an untyped parameter to get around Delphi's strict type checking. The function's header is as follows:

```
function SetToString(Info: PTypeInfo; const Value): string;
```

To give the caller control over the formatting, overloaded functions take additional parameters for prefix, separator, and suffix strings. (Note that *SetToString* uses overloaded functions instead of default parameter values. When you have string parameters, you should use overloaded subroutines. Otherwise, every use of the default string parameter results in a separate copy of the string. Using overloaded subroutines avoids the additional overhead of multiple copies of the same string.)

SetToString is a fairly simple function; it iterates over all possible members of the set, tests the bit for that member, and adds the member's literal representation to the *Result* string if the member is present in the set. Figure 3 shows a first draft of this overloaded function.

```
resourcestring
  sNotASet = 'SetToString: argument must be a ' +
             'set type; %s not allowed';
const
  // Mask to force the minimum set value to be
  // a set element on a byte boundary.
  ByteBoundaryMask = not (BitsPerByte - 1);

function SetToString(Info: PTypeInfo; const Value;
  const Separator, Prefix, Suffix: string): string;
var
  CompInfo: PTypeInfo;
  CompData: PTypeData;
  SetValue: TSet absolute Value;
  Element: 0 ..MaxSet;
  MinElement: 0 ..MaxSet;
begin
  if Info.Kind <> tkSet then
    raise EConvertError.CreateFmt(sNotASet,
      [GetEnumName(TypeInfo(TTypeKind), Ord(Info.Kind))]);
  CompInfo := GetTypeData(Info)^.CompType^;
  CompData := GetTypeData(CompInfo);
  Result := '';
  MinElement := CompData.MinValue and ByteBoundaryMask;
  for Element := CompData.MinValue to CompData.MaxValue do
    begin
      if (Element - MinElement) in SetValue then
        if Result = '' then
          Result := Prefix + OrdToString(CompInfo, Element)
        else
          Result := Result + Separator +
                    OrdToString(CompInfo, Element);
    end;
  if Result = '' then
    Result := Prefix + Suffix
  else
    Result := Result + Suffix;
end;
```

Figure 5: Final version of *SetToString*.

```
procedure
StringToSet(const Str: string;
  Info: PTypeInfo; var Value);
var
  CompInfo: PTypeInfo;
  CompData: PTypeData;
  SetValue: TSet absolute Value;
  MinValue, MaxValue: Integer;
begin
  if Info.Kind <> tkSet then
    raise EConvertError.CreateFmt(sNotASet,
      [GetEnumName(TypeInfo(TTypeKind), Ord(Info.Kind))]);
  CompInfo := GetTypeData(Info)^.CompType^;
  // Initialize SetValue to an empty set. Only initialize
  // as many bytes as are present in the set.
  CompData := GetTypeData(CompInfo);
  MinValue := CompData.MinValue and ByteBoundaryMask;
  MaxValue := (CompData.MaxValue + BitsPerByte - 1) and
              ByteBoundaryMask;
  FillChar(SetValue,(MaxValue-MinValue) div BitsPerByte,0 );
  if CompInfo.Kind in [tkChar, tkWChar] then
    StringToCharSet(Str, CompData, SetValue)
  else
    StringToEnumSet(Str, CompInfo, CompData, SetValue);
end;
```

Figure 6: *StringToSet* divides its work between *StringToEnumSet* and *StringToCharSet*.

The *GetEnumName* function is one of Delphi's standard functions in the TypeInfo unit. It takes an ordinal value, and returns a string literal. If the type is an enumerated type *GetEnumName* returns the enumerated literal. If the type is an integer type, the function calls *IntToStr* to convert the number to a string.

If the set type is a set of characters, though, *GetEnumName* doesn't work correctly. It doesn't test the type kind, so the result is unpredictable — usually an access violation. To solve this problem, write a function that does the same thing, but support characters, integers, and enumerations. Figure 4 shows the *OrdToString* function.

You may have noticed another problem with *SetToString*. Consider the case of an integer or character set whose first member doesn't have ordinal value 0. The function tests the wrong bit in the set. To handle this case, the function must find the correct bit position. If the first member of the set falls after a byte boundary, Delphi saves space and doesn't store the initial empty bytes. *SetToString* must handle this case too.

Finally, the function checks the type info to make sure the type is a set type (type kind of *tkSet*). If not, it raises an exception. Figure 5 lists the final version of *SetToString*.

StringToSet Function

More difficult is the task of converting a string to a set. The string might use any prefix, suffix, and separator characters; it might contain space characters. The string might not be correctly formed. A string can contain enumerated literals, integers, or characters. Characters have multiple representations, as well: quotes (`'x'`), ordinal value (`#13`), or control character (`^M`).

Characters are sufficiently different from integers and enumerations, so two functions are needed. *StringToSet* calls *StringToEnumSet* or *StringToCharSet*, depending on the type kind of the set's component type. Before it converts the string, though, it must initialize the set to empty. The size of the set's value is independent of the type, so *StringToSet* initializes the set to empty by calling *FillChar*. The size of the set depends on the limits of the component type, which must be rounded to byte boundaries. Figure 6 shows the *StringToSet* function.

Starting with *StringToEnumSet* (because it's simpler), the first task is to skip over leading white space characters, then look for a prefix character. Any non-alphanumeric character is allowed as a prefix, but it must be only one character. *SetToString* allows any string, but *StringToSet* must be a little more restrictive to keep it manageable. Then, the function skips more white space and collects an alphanumeric token.

The TypeInfo unit has the *GetEnumValue* function to convert an enumerated literal to its ordinal value. Like *GetEnumName*, it doesn't handle character types. *StringToEnumSet* doesn't handle sets of characters either, so *StringToEnumSet* can call *GetEnumValue*. *GetEnumValue* raises an *EConvertError* exception if the type is an integer type and the string is not a valid integer. It returns -1 for an enumerated type when the name is not valid for the type. *StringToEnumSet* checks for a negative value indicating an error from *GetEnumValue*. It also checks to ensure the ordinal value is in range for the set's type. For any error, it raises an *EConvertError* exception.

StringToEnumSet must find the correct bit position in the set, in the same manner as *SetToString*. Once it finds that position, it sets the bit to one and continues its loop. The next time through the loop, a non-alphanumeric character can be a separator between

```
const
  WhiteSpace = [#0 ..' '];
  Alphabetic = ['a'..'z', 'A'..'Z', '_'];
  Digits = ['0 '..'9'];
  AlphaNumeric = Alphabetic + Digits;
resourcestring
  sInvalidSetString =
    'StringToSet: %s not a valid literal for the set type';
  sOutOfRange =
   'StringToSet: %0 :d is out of range [%1:d..%2:d]';

procedure SkipWhiteSpace(const Str: string;
  var I: Integer);
begin
  while (I <= Length(Str)) and (Str[I] in WhiteSpace) do
    Inc(I);
end;

procedure StringToEnumSet(const Str: string;
  CompInfo: PTypeInfo; CompData: PTypeData;
  var Value: TSet);
var
  ElementName: string;
  Element: Integer;
  MinElement: Integer;
  Start: Integer;
  I: Integer;
begin
  MinElement := CompData.MinValue and ByteBoundaryMask;
  I := 1;
  while I <= Length(Str) do begin
    SkipWhiteSpace(Str, I);
    // Skip the prefix, separator, or suffix.
    if (I <= Length(Str)) and
       not (Str[I] in AlphaNumeric) then
      Inc(I);
    SkipWhiteSpace(Str, I);
    // Remember the start of the set element,
    // and collect the entire element name.
    Start := I;
    while (I<=Length(Str)) and (Str[I] in AlphaNumeric) do
      Inc(I);
    // No name, so skip to the next element.
    if I = Start then
      Continue;
    ElementName := Copy(Str, Start, I-Start);
    Element := GetEnumValue(CompInfo, ElementName);
    if Element < 0 then
      raise EConvertError.CreateFmt(sInvalidSetString,
        [AnsiQuotedStr(ElementName, '''')]);
    if (Element < CompData.MinValue) or
       (Element > CompData.MaxValue) then
      raise EConvertError.CreateFmt(sOutOfRange,
        [Element, CompData.MinValue, CompData.MaxValue]);
    Include(Value, Element - MinElement);
  end;
end;
```

Figure 7: The *StringToEnumSet* function.

set elements. The final loop looks for a trailing non-alphanumeric character as the suffix. If the string cannot be parsed, the function raises an exception. Figure 7 lists *StringToEnumSet*.

Converting a string to a character set is harder, because parsing the characters is more involved. The basic structure of *StringToCharSet* is the same as *StringToEnumSet*, except that each set element must be a character. *StringToCharSet* supports quoted characters and ordinal values after a number sign (#127), which are the same formats used by *SetToString*. Note that Delphi supports one other way to specify characters: A caret followed by a character can be used for control characters (`^M`). However, *SetToString* doesn't use that format, so *StringToCharSet* doesn't either.

```
procedure SaveFont(Font: TFont; Reg: TRegistry);
var
  Style: TFontStyles;
begin
  Reg.WriteString('Name', Font.Name);
  Reg.WriteInteger('Size', Font.Size);
  Style := Font.Style;
  Reg.WriteString('Style',
    SetToString(TypeInfo(TFontStyles), Style));
end;

procedure LoadFont(Font: TFont; Reg: TRegistry);
var
  Style: TFontStyles;
begin
  Font.Name := Reg.ReadString('Name');
  Font.Size := Reg.WriteInteger('Size');
  StringToSet(Reg.ReadString('Style'),
              TypeInfo(TFontStyles), Style);
  Font.Style := Style;
end;
```

Figure 8: Saving a font in the registry.

A quoted character can be a repeated quote (`''''`). An ordinal value can be decimal or hexadecimal (starting with a dollar sign, as per Delphi conventions). The details of parsing characters aren't relevant to this article, so check out Listing One for the full story.

Putting It All Together

Now that you have the *SetToString* and *StringToSet* functions, you need to put them to good use. For example, if you want to store font information in the registry, you can store the font name as a string, the size as an integer, and convert the style to a string, as shown in Figure 8. Because the *Value* parameter is untyped, you cannot pass a property directly to *SetToString* or *StringToSet*, so you must use a temporary variable. If the font style is, say, *fsBold* and *fsItalic*, the registry would contain the string `[fsBold,fsItalic]`.

The Object Inspector already uses a function similar to *SetToString* to display the value of a set-type property. You can write your own property editor for set-type properties and call *SetToString*. You can even let the user type a new set value and call *StringToSet* — something Delphi doesn't currently allow.

Conclusion

The *SetToString* and *StringToSet* functions work with any set of any type, thanks to the wonders of Run-time Type Information. How you use these functions is up to you. ∆

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD in INFORM\00\APR\DI200004RL.

Ray Lischner is the author of *Delphi in a Nutshell* [O'Reilly & Associates, 2000], which contains the full story about Run-time Type Information, and other books and articles about Delphi. He talks about Delphi and programming at conferences and user-group meetings across the country. Ray also teaches Computer Science at Oregon State University.

Begin Listing One — *StringToCharSet*

```
const
  Digits = ['0'..'9'];
  HexDigits = ['a'..'f', 'A'..'F'] + Digits;
  CharBegin = ['#', ''''];
  AsciiChars = [' '..'~'];  // Printable ASCII characters.
resourcestring
  sNotAChar =
    'StringToSet: Not a valid character (%.10s)';
  sCharOutOfRange =
    'StringToSet: Character #%0:d is ' +
    'out of range [#%1:d..#%2:d]';

// Convert a string to a set of character elements.
procedure StringToCharSet(const Str: string;
  CompData: PTypeData; var Value: TSet);
var
  ElementName: string;
  Element: Integer;
  MinElement: Integer;
  Start: Integer;
  I: Integer;
begin
  MinElement := CompData.MinValue and ByteBoundaryMask;
  I := 1;
  while I <= Length(Str) do begin
    SkipWhiteSpace(Str, I);
    // Skip over one character, which might be the prefix,
    // a separator, or suffix.
    if (I<=Length(Str)) and not (Str[I] in CharBegin) then
      Inc(I);
    SkipWhiteSpace(Str, I);
    if I > Length(Str) then
      Break;
    case Str[I] of
      '#':
        begin
          // Character is specified by ordinal value,
          // e.g. #31 or #$A2.
          Inc(I);
          Start := I;
          if (I < Length(Str)) and (Str[I] = '$') then
            begin
              Inc(I);
              while (I <= Length(Str)) and
                    (Str[I] in HexDigits) do
                Inc(I);
            end
          else
            begin
              while (I <= Length(Str)) and
                    (Str[I] in Digits) do
              Inc(I);
            end;
          ElementName := Copy(Str, Start, I-Start);
          Element := StrToInt(ElementName);
        end;
      '''':
        begin
          // Character is enclosed in quotes, e.g. 'A'.
          Start := I; // Save position for error messages.
          Inc(I);
          if (I <= Length(Str)) then begin
            Element := Ord(Str[I]);
            if Str[I] = '''' then
              // Skip over a repeated quote character.
              Inc(I);
            // Skip to the closing quote.
            Inc(I);
          end;
          if (I <= Length(Str)) and (Str[I] = '''') then
            Inc(I)
          else
            raise EConvertError.CreateFmt(sNotAChar,
              [Copy(Str, Start, I-Start)]);
          end;
        else
          // The unknown character might be the suffix. Try
          // skipping it and subsequent white space. Save the
          // original index in case the suffix-test fails.
          Start := I;
          Inc(I);
          SkipWhiteSpace(Str, I);
          if I <= Length(Str) then
            raise EConvertError.CreateFmt(sNotAChar,
              [Copy(Str, Start, I-Start)])
          else
            Exit;
        end;
      if (Element < CompData.MinValue) or
         (Element > CompData.MaxValue) then
        raise EConvertError.CreateFmt(sCharOutOfRange,
          [Element, CompData.MinValue, CompData.MaxValue]);
      Include(Value, Element - MinElement);
    end;
end;
```

End Listing One

By Jon Etheredge

Maintaining State

Are Cookies the Answer to Session Control?

Managing state is an essential underpinning to mission-critical, browser-based applications. These programs need to behave as though they were running in a completely trusted environment. Users must be identifiable, and their actions must remain in context as far as the application is concerned. To do this, the programmer needs to focus on techniques for maintaining state.

The problem, of course, is that applications running in a Web environment are, by definition, stateless. Each time the CGI (Common Gateway Interface) or ISAPI (Internet Server Application Programming Interface) is called by the browser, it treats the call as a new request for information. Essentially "blind" to any previous requests this browser has made, the Web application may need to know certain details that help it determine how these requests are to be handled.

What Is Mission-critical?

The term mission-critical is subject to some amount of interpretation. Certainly, a college student hard-pressed to finish a term paper might consider his midnight Yahoo! searches to be mission-critical, but that sort of activity falls far short of the real definition. When designing Web applications for industrial and government use, the programmer can consider his project to be mission-critical when it meets any one of the following criteria:

- The data being transmitted is strictly regulated by law or common practice, e.g. medical or legal information.
- Unauthorized access to the application might present a risk of commercial or personal loss, either to the organization, the user, or any individual identified in the data stream.
- Inability to access the application causes a work stoppage, corrupts data, or results in financial or personal harm of a grievous nature.

Essential to the definition is the concept of information vulnerability. Whether a reasonable person might expect data from the application to be used in a harmful or unlawful manner is immaterial. Mission-critical applications require a flexible approach to programming that often results in tightly coded, unique products.

Obviously, this is more expensive than off-the-shelf solutions. In part because of the cost factor, the same objection to tight coding is heard in every conference room and every office where Internet projects are planned: "Oh, nobody is really going to break in and steal our data!"

In fact, if US$500 can be made by stealing information from your system, you can go to sleep at night secure in the knowledge that someone, somewhere is working very hard to do just that. If drug-test results can be altered, it's worth somebody's time to attempt it. If children's court records can be opened, someone will make money finding a way to get at the information.

If your company makes its living producing mission-critical applications, a single failure with a single client can put you out of business. No amount of boardroom optimism will ever change that.

What Is Stateless?

Try to imagine a stateless session as a conversation between two individuals who can neither see nor hear the other. One of them remembers everything that has been said (the browser). The other one forgets everything that has been said (the server). The browser asks the server a question and receives a list of possible answers. It then refines the question, based upon the initial response, and asks for more information. Unfortunately, the server has now forgotten the original question.

Session Information
To keep up the "conversation" between server and browser, the browser must send specific information back to the server on each request. This information needs to be unique to each session, and needs to be reliable. It must be immune to guessing, getting lost, or being confused as something else.

The server application will probably want quite a bit of information about each browser request. This might include:

- The age of the current session — No user should be allowed to remain logged in indefinitely. For example, some client locations may have only one browsing machine, and an indefinite session length would allow everybody to use the same session (violating standard security practices).
- The amount of time since the last request — Users occasionally walk away from their workstations in the middle of a session. By limiting this time, the server can control most unwanted data disclosures.
- User rights — The type of information a user could see might be spelled out in access permissions that can be transmitted as part of the session information.
- The address of the browser using this session — While this might not always translate into a valid IP address, it can give important information regarding the physical location of the user.
- The type of browser being used — Referring to this information can help the application determine whether to use JavaScript, how to code HTML, or even whether access is permitted at all.

Much more detail can be preserved, depending upon the specific needs of the Web application. Frequently, far more detail is needed than can be reliably transmitted with each page request.

Using Databases to Maintain State
No matter what method is used to pass session information between browser and server, the validity of that information needs to be checked against a database. Otherwise, the Web application will have no way of knowing whether the session information being transmitted is genuine.

The database can take any form, as long as it holds the session details in a persistent manner. Some ISAPI applications use an internal "database" made up of arrays or lists of objects describing each session. These are only semi-persistent, however, and all users may be forced to log in again if the application crashes or restarts.

Until Delphi 5 came along, ISAPI application developers frequently ran into problems interfacing with databases. As a result, many mission-critical applications ended up coded as CGI instead of ISAPI.

CGI applications are more stable, of course, but they exact a price. First, a separate copy of the CGI must be loaded every time a browser makes a request from the server. Although the cached copy of the CGI loads extremely quickly, the response can be delayed by over a second if it needs to make a database connection. Page production speed is an essential consideration in any Web application, but, since most pages are menus and instructions, there seems to be little reason to wait that additional second while the session information is verified.

Microsoft's Active Server Pages (ASP) use the global.asa database to store session variables about individual users, and use the *Session* object as a means to address these variables. However, if the client database uses InterBase, ASP has trouble coping. Until very recently, ODBC drivers for InterBase were not thread-safe, and had a history of refusing to perform certain operations, such as database inserts.

Session ID
The core piece of information used to maintain state is the session ID. This is a number or string that describes the session in a unique, secure, and reliable manner. It is essentially an index that can be used by the Web application to find specific session information stored in the database.

The session ID must be difficult to guess. If these numbers are issued sequentially or represent indexes into a small base of users, brute-force attacks on the Web application are easier. This usually means resorting to large, random numbers as the session ID.

The Delphi random number generator, however, is limited to 32-bit integers. One solution would be to combine two 32-bit numbers in a composite session ID. For instance, a composite 64-bit random number could be generated with the following code:

```
sSessionID := IntToHex(Random($ffffffff),8) +
              IntToHex(Random($ffffffff),8);
```

This example might produce a session ID looking something like "A23CF8F3." This session ID would be nearly impossible to guess.

Keeping the session ID unique is a bit more involved. The larger the session ID is relative to the installed base of users, the less likely duplicates are to occur. Trusting fate, however, is not a wise strategy in mission-critical applications. If using a *TStringList* of session objects, the application can simply add the session ID to a string in the list. If the string is set to sort automatically, duplicate session IDs can be trapped by setting the *TStringList.Duplicate* property to *dupError*. If the application stores sessions in a database, then the SessionID field of the sessions table should be constrained to use unique values. This way, violations of this constraint can be trapped.

Remote Address Variable
But why generate a unique session ID at all? Why not just use the REMOTE_ADDR environmental variable? In Delphi, that variable is found in the *TWebRequest.RemoteAddress* object property. This returns the IP address assigned to each browser. Because these are unique by definition, they seem to be ideal candidates for use as session IDs.

However, the browser's IP address presents one major problem that severely limits its usefulness as a session identifier. Users working behind address-translating firewalls or application proxy servers may not be able to send their actual identity across to the Web application. This is a common problem when dialing in through an ISP. The user's IP address might be translated into something like "philmax1-p75.mississippi.net." Because the *RemoteAddress* property is unreliable in large systems, it should never be used as the session ID.

Whatever form the session ID takes, it needs to be sent between the browser and the Web server on every request. The method by which this information is sent deserves close inspection.

Session Information in Cookies
Cookies help track state information by recording essential data in small files maintained by the browser on the client's hard drive. ASP

```
procedure TWebModule1.WebModule1WebActionItem1Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  tslCookie: TStringList;
begin
  tslCookie := TStringList.Create;
  tslCookie.Add('USERID=JME');
  Response.SetCookieField(tslCookie,'mydomain.com',
                          '/scripts',Now,False);
  Response.Content := 'Cookie sent!';
  tslCookie.Free;
end;
```

Figure 1: Sending a cookie from a Delphi Web application.

uses cookies as its principal means of passing session identification variables between the server and the browser. Cookies are equally simple to use in Delphi, although more knowledge of the underlying code is required. To send a cookie from a Delphi Web application is very simple, as shown in Figure 1.

This sends a cookie to the browser along with the response stream. Whenever the browser requests information from a Web application located in the http://mydomain.com/scripts directory, this cookie — if it's on the hard drive — will be sent to the server as part of the request stream. The operative phrase is "if it's on the hard drive." The third parameter in the *SetCookieField* method contains the value "Now." You might interpret this to mean that the cookie expires immediately, but it's not that easy. "Now," in cookie terms, may not actually be *now*.

As an experiment, set up Netscape to prompt you before accepting a cookie. Then write a test CGI in Delphi that sends a cookie set to expire at "Now." Finally, save your program in a scripts directory on your workstation (running PWS or IIS), and load the CGI. If your workstation is running under Central Daylight Savings Time (this is GMT [Greenwich Mean Time] minus six hours), you'll be told that the cookie expired nearly 30 years ago!

The problem is that the cookie always assumes that the server was running under GMT. Once on your browser, the cookie does some math. It figures out what the GMT time is relative to your local time, and then sets itself up to expire then. If your server was set up under Central Daylight Savings Time, the cookie is actually running a little late — about six hours worth.

It takes a little work, but once you have a firm handle on how to set up the cookie expiration date, you can explicitly limit the session time by limiting the lifespan of the cookie. Your CGI only needs to read the contents of the cookie, and if there are none, force the user to log in again. Reading a cookie is far easier than sending one; for example:

```
procedure TWebModule1.WebModule1WebActionItem2Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content:='<html><body><h1>COOKIE TEST</h1><hr>'
    + 'USER ID = ' + Request.CookieFields.Values['USERID']
    + '</body></html>';
end;
```

Some cookies never get stored on the browser's hard drive. These are known as "session cookies" because they only stay alive as long as the browser is on. To set up a session cookie, the expiration date needs to be omitted from the cookie. In Delphi, this is done by setting the date string to "-1" instead of "Now". Cookies of this type can be used to help determine if a user has shut down their browser.

If more information needs to be exchanged between the browser and the server, the *TStringList* object used to set the cookie should have additional Name=Value pairs added to it. Other than that, the code in the *TWebResponse.SetCookieField* method is the same. The overall effect, however, is quite different.

Drawbacks to Using Cookies

For each Name=Value pair sent to the *SetCookieField* method, an additional cookie is sent in the response stream. More cookies are sent as the session data set becomes more complex. This can eventually overrun the maximum cookie limit for a domain (20 cookies), with serious consequences. The oldest cookie from that domain will be dropped. Unfortunately, if all the cookies have the same date, they could all be dropped. In a best-case scenario, some session information will be lost. At worst, the cookies won't work at all, and might even crash the browser.

One way to get around the 20-cookie domain limit is to send all the session information as a single string, delimited somehow. Separating fields with a special character (& for instance) will allow you to cram more information into a single cookie. There are only two limitations to this technique. One, you can't have more than a single "=" (equal) sign anywhere in the string. Otherwise, the *SetCookieField* method will split the information into two cookies. And two, there is a size limit to cookies; they cannot exceed 4,096 characters. If your session information includes complex SQL strings, for instance, you can end up with an invalid cookie.

There are other problems with cookies that deserve a closer look. To function, cookies depend upon path specificity. Cookies are sent back to the server if the browser's URL matches the domain/path specified in the cookie. A cookie set to respond to "/scripts/AnyCGI.Exe", for example, couldn't be redirected to "/scripts/PassChange.Exe". To allow redirection, the path specification in this example would have to be changed to "/scripts/". Unfortunately, that cookie would also be sent to "/scripts/LaundryList.Exe", and might overwrite information from properly authorized cookies.

Cookies are vulnerable to a more insidious problem as well: They can be spoofed. Programmers can use the "domain override bug" to set up a cookie domain field of "anywhere.com..." . When users holding this cookie hit the site at "nowhere.com.../scripts/Test," the cookie is sent. This gets around the domain privacy model inherent in cookies, and is a function of the browser being used. Internet Explorer doesn't store this cookie in a persistent state, running it instead as a session cookie. Netscape runs the cookie normally.

The domain override bug only affects cookies carrying domain names, not IP numbers. And its use as a hacking tool is highly questionable. Nevertheless, because of well-publicized, and often misinterpreted flaws in cookie design, many organizations have installed firewalls and proxy servers that have the ability to strip cookies from the response stream.

If your Web application requires cookies, its design may require clients to change their organizational MIS policies before they can use your program. How difficult can this be? Until recently, AOL users couldn't receive cookies at all. The doctrine of requiring cookies in mission-critical applications, therefore, may provide an insurmountable marketing challenge. The negative impact on the developer's long-term income should deter such an approach.

Managing State with Forms

Before there were cookies, there were "Hidden" fields. These do not show up in your browser forms, but are passed along with server requests anyway. There are two ways to send form information back to the Web server: use the *Get* and *Post* methods.

The *Get* method takes the Name=Value pairs from each form input field and appends them to the URL. They're then passed into the Web application in the QUERY_STRING environmental variable. In Delphi, *Get* variables are passed through the *TWebRequest.Query* object. One principal disadvantage of using the *Get* method is that the query string is displayed in the browser's address window. Even passwords, normally protected from eavesdropping by hiding them in "password-type" fields, are plainly visible if sent with the *Get* method.

The *Post* method sends the information in a separate data stream, which the Delphi Web application reads through the *TWebRequest.Content* object. The following HTML lines, for example, will send the form information to the server using the *Post* method:

```
<Form Action="/scripts/AnyCGI.Exe" Method="Post">
  <Input Type="HIDDEN" Name="UserID" Value="JME">
</Form>
```

The Web application will decode this form as `Request.ContentFields.Values['UserID']='JME'`. It's just as easy to deal with on the receiving side as information in a cookie, but requires considerably more planning to set up. Some programmers consider hidden fields less secure than cookies because the View Source browser command lets users see what information is stored there.

Being able to see the contents of hidden fields is not a serious drawback, however. The only information that should be absolutely required would be the unique session ID. Anything else needed to track this session should be stored in a persistent database on the Web server.

Using URL Variables

One major benefit of using the *Get* method is the fact that the QUERY_STRING variable can be set without using a form at all. Web applications can append information of almost any type to the end of the URL. This technique can be used to pass session-tracking information between Web applications, or even between Web servers.

Here's what that paragraph means in practice. Suppose you have an application that presents the user with a list of options, one of which is "Change Password." Your password modification program, however, is in a separate CGI. What your code needs to do is forward the request to the new program, using the following Delphi code:

```
with Request.ContentFields do
  if Values['Action'] = 'Change Password' then
    Request.SendRedirect('/scripts/NewPassword.Exe');
```

The trouble is that this code sample won't work. Information passed with the *Post* method can't be redirected (a limitation of the HTTP specification). The entire *TWebRequest.Content* object will be discarded. Of course, you could recode all your Delphi Web applications to use the *Get* method, but it's a lot easier to simply append the session ID to the redirection request:

```
with Request.ContentFields do
  if Values['Action'] = 'Change Password' then
    Request.SendRedirect('/scripts/NewPassword.Exe?' +
                         Values['SessionID']);
```

The new application can then look at the value of the *TWebRequest.Query* property. It will contain a single number: the session ID.

There are limitations associated with appending data to the URL, regardless of whether you put it there in the code or use the *Get* method. The most serious of these is the fact that you can send a maximum of only 255 characters. Send more, and the browser could crash. This is one of the major reasons to avoid using the *Get* method to maintain state in applications that pass a lot of SQL strings between multiple pages. It's far more reliable to store your session variables in a database, and simply pass the session ID.

This technique removes the limitation requiring you to always send information in a form. By appending the session ID to the URL, your Web pages can maintain state inside links, as well. For example, to link to a special CGI, your HTML code might look like this:

```
<A HREF="/scripts/NewCGI.exe?FF2B87A3>
```

In this example, the session ID, "FF2B87A3", is sent as the only variable in the query string. The Web application seeing this in the *TWebRequest.Query* property would typically check to see whether a session exists for this ID, and would either continue processing, or halt (if the session ID didn't match).

Sending Information with JavaScript

Java applets, ActiveX controls, and cookies can all be blocked with appropriate firewall or proxy server applications. This is because any element that exists outside of the HTML document itself can be stripped away before the browser ever sees it. JavaScript, on the other hand, only exists inside the HTML document. It always gets delivered to the browser.

Of course, individual users could set their browsers to reject JavaScript. This is less of a problem than firewall-level blocking, as the programmer doesn't have to contend with organizational policies in order to make the product function.

Just to be on the safe side, however, mission-critical applications should avoid the use of JavaScript in essential functions (such as form submission buttons) wherever possible. Always have a backup method to submit forms.

That being said, JavaScript is still one of the more useful of the Web programming languages. If your Web application uses graphical buttons, JavaScript is the only way to use those images to submit forms.

Conclusion

There are problems that programmers can control, and some they can't. They can control the amount of information exchanged in session tracking. They can control the method of transfer. They can control the storage medium for session details.

The amount of data sent between the browser and Web server should be kept to a minimum. The session information exchange should normally consist of a unique, secure identifier, and nothing else. Everything the application needs to know about the session should be retrieved from storage.

Data sent between the browser and the Web server should be as invisible as possible, but needs to match the intent of the data. For instance, password forms should never use the form *Get* method, because the password and user ID will eventually end up in the browser history for anyone to see. The session ID can be sent openly without fear of spoofing if it's properly designed. Forms, JavaScript, and URL appending can all be useful transfer methods, as long as their individual limitations are understood.

The session information storage medium needs to be fast and robust. Keeping session information in a list of objects on the DLL is fast and easy, but suffers from volatility. The contents of this list should be committed to an actual database before shutting down the Web server, or all your users will have to log in again.

There will always be problems programmers can't control. Database drivers can interfere with one another, or with the correct functioning of the application. Web site design requirements may require a frames-based approach to forms. Client functional requirements might dictate the need to code exclusively in CGI or to support outdated browsers.

Perhaps the most restrictive problems involve organizational policies that limit user access across the Internet. These doctrines may require firewall and proxy filters that exclude cookies, ActiveX, and Java applets. Unless the programmer has absolute authority over every aspect of user connectivity, filters such as these must be assumed to be in place somewhere. For that reason alone, cookies should never be the exclusive session-tracking method in any mission-critical application. ∆

Jon Etheredge is a Senior Application Developer for Vision Software in Meridian, MS, with over 15 years experience in military, medical, and legal software systems. Vision Software provides consulting services, data collection, time and attendance, maintenance management, Web-based database solutions, and custom applications for industrial and government clients across the United States. Contact Vision Software at (800) 464-1244, or visit their Web site at http://www.vsnsoft.com.

By Chris Austria

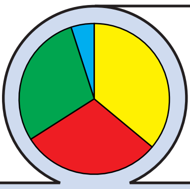
Readers Choice Awards 2000

Y2K Bug, What Y2K Bug?

With the Y2K brouhaha behind us, it's time to announce your picks for the best third-party Delphi development tools for 1999. The dedicated companies that created these products obviously decided to eschew the millennial hysteria, focusing on the future instead. Reflecting the spirit of the greater Delphi community, these companies continued to work hard to fashion tools that make Delphi developers' lives easier, and to make them more productive and efficient than ever.

The release of Delphi 5 also provided a new opportunity for many of these third-party companies to provide new and improved versions of their various tools. The Delphi third-party market is ever changing, reacting to the many forces in the industry. This year's ballot reflects those changes in the form of new products and re-shaped categories. An interesting mixture of old and new can be found in many categories, resulting in an exciting dynamic we've come to expect in the Delphi tools community.

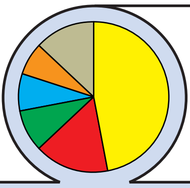
Best Accounting Package



🟨 AdaptAccounts
🟥 Bravo
🟩 Accounting for Delphi
🟦 Other

I predicted a close call in this category when I reported last year's winners, and I wasn't disappointed. The three top contenders all made respectable showings, with AdaptAccounts by Adapta Software the clear winner with 36 percent of the votes. Second place was not-so-clear, and we're considering Bravosoft's Bravo, and ColumbuSoft's Accounting for Delphi, tied for second place with 30 and 29 percent of the vote, respectively. This is definitely a category to keep an eye on next year.

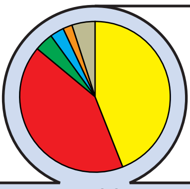
Best Add-in



🟨 CodeRush
🟥 Multi-Edit
🟩 StarTeam
🟦 ClassExplorer
🟧 CodeWright
⬜ Other

Part of Best Add-in/Library last year, this category now consists solely of Delphi add-in products. A more clear-cut race was just what Eagle Software's CodeRush needed to take first place; it flew by the rest of the pack, collecting 47 percent of the votes. The next closest competitor is Multi-Edit from American Cybernetics, which collected 16 percent (making it the most popular third-party editor). CodeRush placed second last year, a significant 21 percent behind SysTools from TurboPower, which is now in the Best Library category. What a difference a year — and a well-defined category — makes.

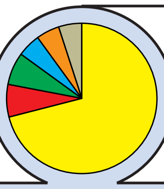
Best Book



🟨 *Mastering Delphi 5*
🟥 *Borland Delphi 5 Developer's Guide*
🟩 *Tomes of Delphi: Win32 Database Developer's Guide*
🟦 *Tomes of Delphi: Win32 Graphics Programming*
🟧 *Delphi Developer's Guide to OpenGL*
⬜ Other

One of the few categories left untouched, Best Book offered a slightly smaller pile of books to select from this year. From the seven in this year's ballot rose two clear winners. First and second place honors go to *Mastering Delphi 5* [SYBEX] by Marco Cantù, and *Borland Delphi 5 Developer's Guide* [SAMS] by Steve Teixeira and Xavier Pacheco, respectively. *Mastering* garnered 44 percent, and *Borland Delphi* 42 percent. Both were way ahead of the group, with the next leading book collecting 4 percent of the votes.

Best Charting/Mapping Tool

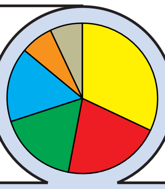


- 🟨 teeChart Pro
- 🟥 ProEssentials
- 🟩 MapObjects
- 🟦 Graphics Server
- 🟧 Olectra Chart
- 🟫 Other

Another category change was deemed necessary here. Last year's Best Charting/-Imaging category has become Best Charting/Mapping, with Best Imaging Tool now its own category. teeChart Pro from teeMach charted an incredible 71 percent of the votes in this category, with the closest competition, Gigasoft's ProEss-entials, taking only 7 percent.

The need to reshape this category was apparent when we looked at last year's results, where teeChart Pro (a charting product) edged out ImageLib Pro (an imaging product) 41 to 36 percent. Although we have a repeat winner this year, ImageLib Pro now has a shot at taking first place in a distinct imaging category. (You'll soon see whether it does.)

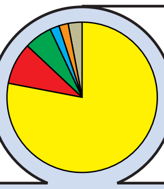
Best Database Engine



- 🟨 Apollo
- 🟥 FlashFiler
- 🟩 Advantage Database Server
- 🟦 DBISAM
- 🟧 TOPAZ for Delphi
- 🟫 Other

Although new, this category includes some familiar players; Advantage Database Server, FlashFiler, Apollo, DBISAM, and TOPAZ have made appearances in previous Readers Choice Awards. Placing them in a more function-specific database category provided good competition this year. Coming out on top was Apollo from Vista Software (previously from Luxent Software & Webworks) with a hefty 32 percent. FlashFiler from TurboPower wasn't far behind with 21 percent.

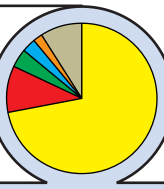
Best Communications Tool



- 🟨 Async Professional
- 🟥 WebHub
- 🟩 IP*Works! Delphi Edition
- 🟦 Visual Voice
- 🟧 PowerTCP Internet Toolkit
- 🟫 Other

Now dubbed Best Communications Tool, this is simply last year's Best Connectivity Tool category renamed. This category's name is now more precise, and distinct from the new Database Connectivity category (coming up next), but it didn't change the outcome of the voting. The top four winners are identical to last year's, and show the same voting pattern. Again, Async Professional from TurboPower Software muscled its way ahead of the pack with an amazing 78 percent of the votes, up 10 percent from last year. In fact, Async Pro has dominated this category since the 1997 Readers Choice Awards. Next is HREF Tools' WebHub, gathering 9 percent of the votes.

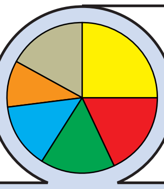
Best Database Tool



- 🟨 InfoPower
- 🟥 Rubicon for Delphi
- 🟩 SQL Navigator/TOAD
- 🟦 AdHocery
- 🟧 Brickhouse Object Architecture
- 🟫 Other

As we weeded out the more specialized database products, this category's participants are fewer — and more competitive. The winner, Woll2Woll Software's InfoPower, took advantage of the situation and powered its way to the top with an incredible 72 percent of the votes, compared to a mere 8 percent last year. Talk about coming from behind! Second place went to Rubicon for Delphi, from Tamarack Associates, which garnered 10 percent of the votes. Will someone bridge this huge gap next year, or will InfoPower simply pull further away from the pack?

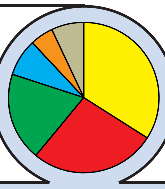
Best Database Connectivity



- 🟨 IB Objects
- 🟥 ASTA
- 🟩 Direct Oracle Access
- 🟦 ODBC Express
- 🟧 Titan
- 🟫 Other

There's been a marked increase of database-related tools, so one category was no longer sufficient. Now there are three: Connectivity, Engine, and Tool. In its first appearance in the Readers Choice Awards, the Database Connectivity category provided an exciting turnout. First place clearly goes to Jason Wharton's IB Objects, with 25 percentage points. Jason is followed by ASTA from ASTA Technology Group, which gathered 18 percent. The next three products came in with 16, 14, and 10 percent, respectively, and promise to provide exciting competition in the years to come.

Best Help-authoring Package

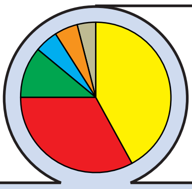


- 🟨 HelpScribble
- 🟥 RoboHELP
- 🟩 ForeHelp
- 🟦 DotHLP/DotCHM
- 🟧 Time2HELP
- 🟫 Other

At some point, everyone who uses software needs a little (or a lot of) help. The participants in this category know this, and devote a lot of energy helping developers help their end users. Last year, HelpScribble from Jan Goyvaerts (JGsoft) helped itself to the top title, barely edging out Blue Sky Software's RoboHELP by 1 percent.

This year, HelpScribble gains some breathing room, compiling 34 percent, followed by RoboHELP, with 27. Worthy of mention is this year's and last year's third-place winner, ForeHelp from ForeFront, which never seems far behind. This category never fails to provide an exciting finish, and I'm positive next year will offer more of the same.

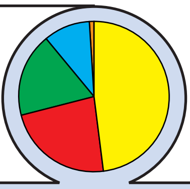
Best Imaging Tool



- 🟨 VisImage ActiveX Pro
- 🟥 ImageLib Corporate Suite
- 🟩 LEADTOOLS Imaging Toolkit, et al.
- 🟦 ImagN'
- 🟧 ImagXpress
- ⬜ Other

As promised, the imaging products now have their own arena to battle in. VisImage ActiveX Pro from Visionary Solutions is the clear winner this year, gathering a hefty 42 percent of the votes. Also in the frame, however, is SkyLine Tools Imaging's ImageLib Corporate Suite, with a very respectable 33 percent of the votes. I'd keep an eye out for these two, as I'm sure they're each picturing ways to take the first-place honors home next year.

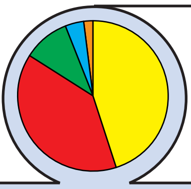
Best Modeling/CASE



- 🟨 CDK
- 🟥 Propel
- 🟩 WithClass!
- 🟦 Pro-Analyzer
- 🟧 Other

The Best Modeling/CASE winner gets bragging rights as the first champ in a new category. Those rights go to CDK from Eagle Software, which compiled 48 percent of the votes, a comfortable 25 percent more than Nevrona Designs' Propel, with 23. Although in a different category, the products in the Best Modeling/CASE category are previous Readers Choice participants and have always provided great competition in other categories in the past, so we can expect no less from them in the future.

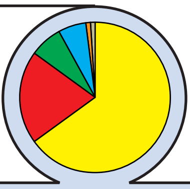
Best Installation Package



- 🟨 Wise Installation System
- 🟥 InstallShield Express
- 🟩 Youseful
- 🟦 InstallFromTheWeb
- 🟧 PC-Install

Your good software means nothing if you can't get it deployed properly on users' computers. The products in this category work hard to make things work for you, and the fight to the top was a nail-biter this year. Last year provided stiff competition as InstallShield Express from InstallShield Software beat out Wise Installation System from Wise Solutions. This year, however, Wise Installation System, with 45 percent of the votes, gained enough ground to clearly pass InstallShield Express, which garnered 39 percent of the votes.

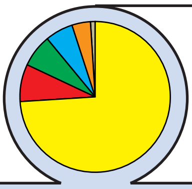
Best Reporting Tool



- 🟨 ReportBuilder
- 🟥 ReportPrinter Pro
- 🟩 ACE Reporter
- 🟦 Shazam Report Wizard
- 🟧 PrintDAT!
- ⬜ Other

Last year, when ReportBuilder from Digital Metaphors came out of nowhere and decidedly took the title from two-time winner ReportPrinter Pro, I knew a great rivalry was in the mix. This year, ReportBuilder again dominated the ballot with a sizable 65 percent of the votes in this category. Nevrona Designs' ReportPrinter Pro must again settle for second, with a respectable 20 percent. If Delphi Readers Choice history is any indication, picking the Best Reporting Tool next year won't be a black-and-white decision.

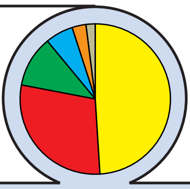
Best Library



- 🟨 SysTools
- 🟥 LockBox
- 🟩 ExpressBars Suite
- 🟦 Protection Plus
- 🟧 NWLib
- ⬜ Other

For the first time in the history of the Readers Choice Awards, first and second place in one category go to a single company, in this case TurboPower Software. Despite competing with a sister product, the company's SysTools collected a commanding 74 percent, and its LockBox locked in 8 percent of the votes. Congratulations to TurboPower for their stellar performance.

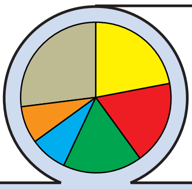
Best Testing/Debugging Tool



- 🟨 Memory Sleuth/Sleuth QA Suite
- 🟥 CodeSite
- 🟩 NuMega BoundsChecker
- 🟦 QTime
- 🟧 reAct
- ⬜ Other

In the end, we all know everyone makes mistakes, including developers. Hence, we embarked on a search for the Best Testing/Debugging Tool, this year in its own category for the first time. In first place is Memory Sleuth/Sleuth QA Suite, from no other than TurboPower Software, compiling 49 percent of the votes. Second place goes to Raize Software Solutions' CodeSite, with a respectable 29 percent. How better to test and debug your code than with these winners?

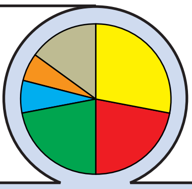
Best Training



🟨 InfoCan Management
🟥 Blue Star Training & Software
🟩 Database Programmers Retreat
🟦 GenoTechs
🟧 Keystone Learning Systems
🟫 Other

I can always count on the Best Training players to provide a good match. This year's fight to the top left three standing in close proximity. For the third year in a row, InfoCan Management managed to out-train the rest, gathering 22 percent of the votes. Not far behind, however, are Blue Star Training & Software, with 18 percent of the votes, and Database Programmers Retreat with 17. Blue Star wasn't among the top five finishers last year, and if this second-place showing is any indication of what this company is capable of, look for them next year.

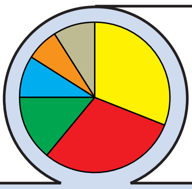
Best VCL Component Set



🟨 Orpheus
🟥 Raize Components
🟩 1stClass
🟦 LMD-Tools
🟧 WPTools
🟫 Other

As if one Best VCL category spin-off wasn't enough, TurboPower powered its way through this category as well. With a solid 28 percent of the possible votes, TurboPower's Orpheus finished 6 percentage points ahead of Raize Software Solutions' Raize Components and Woll2Woll Software's 1stClass, each with 22 percent of the votes. Yep, it's another tie, this time for second. All three are worthy and popular products, and I predict another close encounter next year.

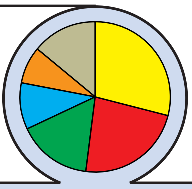
Best VCL Component



🟨 Abbrevia
🟥 ExpressQuantumGrid
🟩 TopGrid
🟦 TSyntaxMemo
🟧 GTSizer
🟫 Other

The previous Best VCL category is now split in two. This proved to be a sound decision, as this, the Best VCL Component category, afforded the tightest finish out of all the categories in the ballot. Amassing 31 percent of the votes, TurboPower Software's Abbrevia barely edged out Developer Express' ExpressQuantumGrid, with 30 percent of the votes. This one's too close to call — a statistical tie — with third place honors going to Top Support's TopGrid.

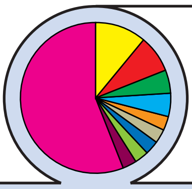
Best Utility



🟨 Xceed Zip
🟥 DynaZIP
🟩 Barcode Suite
🟦 Runtime Designer for Delphi
🟧 Olectra Resizer
🟫 Other

Mostly due to the re-shaping of the ballot this year, none of last year's top five contenders are even in this year's slimmer Best Utility ballot. As a result, a fresh set of competitors emerged, each attempting to prove its worth as the Best Utility for Delphi. Zipping up the top two spots are Xceed Software's Xceed Zip, with 29 percent of the votes, and DynaZIP from InnerMedia, with 23 percent. In such a heterogeneous category, it's worth mentioning that third place went to Barcode Suite (from SkyLine Tools Imaging), making it the most popular bar-coding tool.

Product of the Year



🟨 ReportBuilder	🟫 Async Professional
🟥 Apollo	🟦 Advantage Database Server
🟩 InfoPower	🟩 DBISAM
🟦 HelpScribble	🟪 ExpressQuantumGrid
🟧 CodeRush	🟪 Other

Once again, we wrap up with the single-most coveted award in the Readers Choice Awards. For a year, the winner in this category gets bragging rights as the product that has proven itself the most popular product by Delphi developers. That's quite an honor.

This year's winner is a repeat performer, ReportBuilder from Digital Metaphors. This time they managed to build a 3-percent lead, garnering 11 percent of the votes, compared to last year's 2-percent lead over TurboPower's Orpheus. Another surprise was that Orpheus wasn't in the top nine finishers this year. Second place instead goes to Vista Software's Apollo, which wasn't in the top finishers last year, with 8 percent of the votes. It just goes to show what a year of perseverance and hard work will do.

Company of the Year

There is no Best Company category on the ballot, but the winner is clear nevertheless. Special congratulations are in order for TurboPower Software. Just in case you haven't been keeping track, a TurboPower product placed first in five categories — five! — and second in two. This feat is completely unprecedented in the history of the awards.

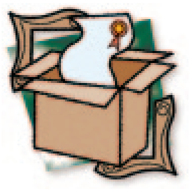
Conclusion

Developers never tire of seeing new and improved products for their favorite development environment, and Delphi's third-party offerings in 1999 have fulfilled their expectations. While the rest of the world held its breath awaiting the start of the two-thousandth year (actually the 1999th, but who's counting), the Delphi community looked forward, getting a head start on a new and better year. Thanks to all the participants in this year's awards for their dedication to Delphi and their vision of the future of Delphi development. Without their innovative products and continued dedication to improvement, there would be no Readers Choice Awards. The existence of such a diverse and powerful third-party tools community is a testament to the appeal and endurance of Delphi.

Finally, a thank you to all of you readers who took the time to visit our Web site and vote for your favorite products. (By the way: For the first time, *every* vote was made at our Web site; we received no faxed or snail-mail ballots.) Your opinions not only help the companies who look for feedback so they can continue to improve their products, your votes also help other Delphi developers select the products they need to be as efficient and productive as possible. See you all next year. ∆

For information on contacting the winners, visit http://www.DelphiZine.com, filename di200004ca_f.

Chris Austria is Products Editor at *Delphi Informant Magazine*, and can be reached via e-mail at caustria@informant.com.



By Bill Todd

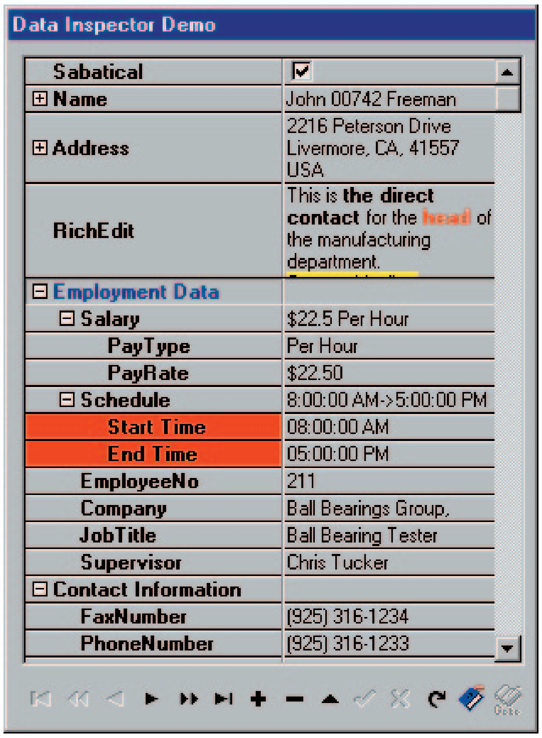
InfoPower 2000

A Must-have Delphi Component Suite Gets Better

Woll2Woll Software's InfoPower gets better with every new release, and InfoPower 2000 for Delphi 4 and 5 and C++Builder 4 is no exception. Let's start by looking at the new features before reviewing the whole InfoPower component suite.

The architecture of the InfoPower components has been changed to support any *TDataSet* descendant. In the past, if you wanted to use an InfoPower data-aware component, such as the *TwwDBGrid*, you had to use the InfoPower dataset and data source components to provide the data. Now you can use any InfoPower data-aware control with any dataset and data source components, as long as the dataset is a descendant of *TDataSet*. This means that the InfoPower controls now work with all of the standard Delphi dataset components, including the ADO Express and InterBase Express components. Because the dataset components used by virtually all third-party database engines are descendants of *TDataSet*, you can now use InfoPower components with them as well.

Another small change that affects database engine compatibility is that customer picture masks you create at design time are no longer stored in a Paradox table. Instead, custom picture masks are now stored in InfoPowerMasks.ini so you don't need to have the BDE available to work with picture masks.



Data Inspector Demo

Sabatical	✓
⊞ Name	John 00742 Freeman
⊞ Address	2216 Peterson Drive Livermore, CA, 41557 USA
RichEdit	This is **the direct contact** for the head of the manufacturing department.
⊟ Employment Data	
⊟ Salary	$22.5 Per Hour
PayType	Per Hour
PayRate	$22.50
⊟ Schedule	8:00:00 AM->5:00:00 PM
Start Time	08:00:00 AM
End Time	05:00:00 PM
EmployeeNo	211
Company	Ball Bearings Group,
JobTitle	Ball Bearing Tester
Supervisor	Chris Tucker
⊟ Contact Information	
FaxNumber	(925) 316-1234
PhoneNumber	(925) 316-1233

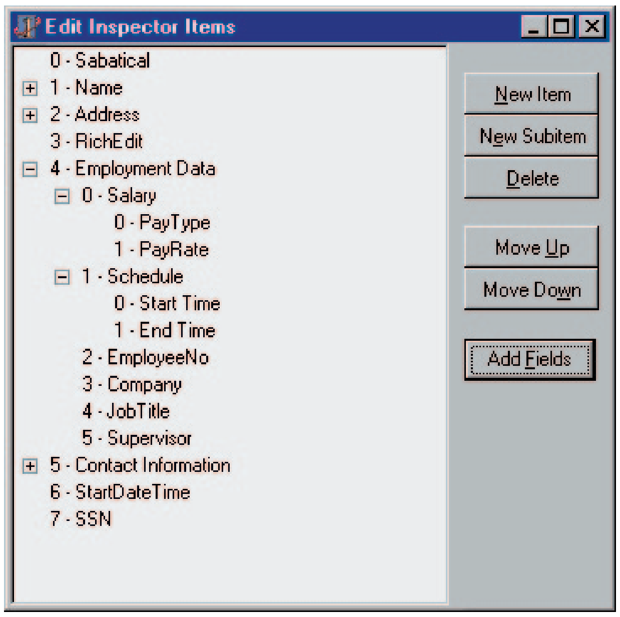
Figure 1: The Data Inspector.

Major Enhancements

TwwDataInspector. The major new component in InfoPower 2000 is the *TwwDataInspector*. The Data Inspector is to data what the Delphi Object Inspector is to objects and properties. Figure 1 shows the Data Inspector displaying data from a single table. The Data Inspector lets you group data from one or more tables hierarchically, and gives users the ability to expand the various categories to see progressively more detail. Each item in the tree has its own *DataSource* and *DataField* properties, making it an excellent tool for displaying one-to-many relationships to any depth while using little screen real estate. The built-in Navigator makes browsing and editing data easy and familiar for users. The Data Inspector can also be used as an unbound control so you're not limited to displaying data from a database. This makes it an excellent tool for setting configuration options or object properties within your application.

Setting up the Data Inspector at design time is easy thanks to its Items Editor. Simply double-click the Data Inspector to display the editor shown in Figure 2. Using the Items Editor, you can add additional rows to the Data Inspector at any level. You can also select any item and set its properties or event handlers using the Object Inspector. Like all of the InfoPower components, the Data Inspector and its items have a rich set of events so you can add custom behavior to any user action. Because you can embed any of the InfoPower data-aware controls or any of the controls from Woll2Woll's 1st Class component suite in the Data Inspector, you can display data using any component that is appropriate. You can even display rich text with all of its formatting.

Transparency. Another major new feature of InfoPower 2000 is transparency and custom framing support in all of its components. This is one of those cases where a picture is truly worth a thousand words (see



Edit Inspector Items

```
0 - Sabatical
+ 1 - Name
+ 2 - Address
  3 - RichEdit
- 4 - Employment Data
    - 0 - Salary
          0 - PayType
          1 - PayRate
    - 1 - Schedule
          0 - Start Time
          1 - End Time
      2 - EmployeeNo
      3 - Company
      4 - JobTitle
      5 - Supervisor
+ 5 - Contact Information
  6 - StartDateTime
  7 - SSN
```

Figure 2: The Data Inspector Items Editor.

Figure 3). All InfoPower controls now have a *Transparent* property, which allows the background to show through when set to True. All of the components also have a *Frame* property, which controls the appearance of the frame when the component has focus, and when it does not.

The check form in Figure 3 was built by dropping a *TImage* control on the form and setting its *Align* property to *alClient*. The *TImage* contains the background bitmap for the check. All the controls on the form have their *Transparent* property set to True to allow the bitmap to show through. In addition, all the edit controls have their *Color* property set to *clWindow* so they will appear white when they have focus. The check number in the upper-right corner shows how the appearance of the controls changes when they have focus. This is accomplished by setting the frame property so that all four sides of the frame are visible when the component has focus, but only the bottom edge is visible when the component does not have focus. Although the check number uses a simple box frame style, you can also choose bump, raised, lowered, or etched for a more sculptured look. The *Transparent* and *Frame* properties make it easy to create impressive forms that look like their paper counterparts.

TwwDBGrid. One of the stars of the InfoPower suite since version 1 is the *TwwDBGrid*, and it has been enhanced in two important ways. First, the grid now has the ability to stream its display settings to or from an INI file or the registry at run time. This lets you give users the ability to change the grid's appearance by changing column widths and column order, and save their settings so the grid will look the same the next time they start your application. Another problem shared by all grids is the tradeoff between making columns wide enough to show the full content of long text fields and keeping them narrow enough to show all the fields without scrolling horizontally. InfoPower's solution is the option to place the mouse cursor on a cell and have the full text appear in a pop-up hint window.

The InfoPower grid has always had the ability to embed its own check box and combo box controls in the grid, but it now includes the ability to embed many of the components from Woll2Woll's 1st Class product as well. Other features of the grid include displaying memo and rich text fields in the grid editing memo or rich text fields in a pop-up editor; multi-line rows; word wrap; turning off the row and/or column lines; using the column headings as pushbuttons; treating the Enter ↵ key as Tab; leaving the grid if Tab ⇆ is pressed in the last column; footer cells for totals; picture mask support; displaying graphics in cells or column headings; defining columns that remain in view during horizontal scrolling; editing lookup and calculated fields; and setting the color, font, and alignment of the column headings. Even this list barely scratches the surface. To appreciate the power of the grid, you need to read the chapter on the grid in the InfoPower *Developer's Guide*, which is over 30 pages long.

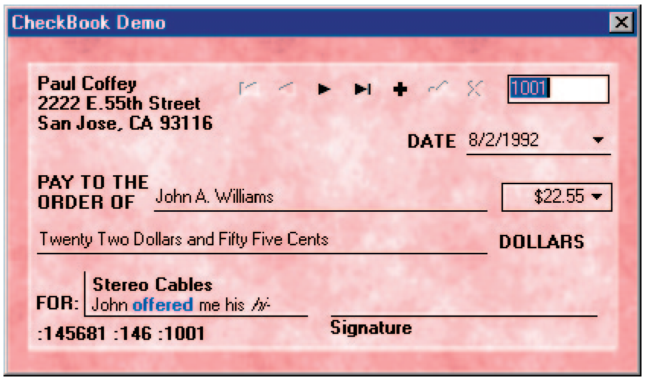
TwwDBRichEdit. The *TwwDBRichEdit* component is a powerful pop-up word processor for creating and editing rich text. It now offers the option to use the Microsoft Word spell checker and grammar checker if Word is installed on the user's PC. The dictionary and spelling options the user has set in Word are used automatically in InfoPower. Users can now set the background color of highlighted text in the rich edit component's word processor, and the word processor now uses bitmapped menus. Several new events have also been added to make it much easier to customize the behavior of menu items in the word processor.

Some of My Favorites

TwwDBCombo. The InfoPower suite includes too many components to describe in detail, so I'll focus on my favorites. The *TwwDBCombo* box offers many features that the Delphi equivalent doesn't. Perhaps the most valuable of these is the ability to enter pairs of values, one of which is displayed in the combo box and the other is actually stored in the database. You can also turn on Quicken-style incremental searching, which many users like. The *TwwDBComboDlg* component looks almost identical to a combo box, but the button contains an ellipsis instead of a down arrow. You use this component to display your own custom dialog box to aid the user in editing a field value.

TwwDBDateTimePicker. The *TwwDBDateTimePicker* provides a drop-down calendar to make choosing a date easy. To enter the current date, simply tap the spacebar. At the end of the date, you can tap the spacebar again to enter the current time. The control will also optionally display week numbers and you can control whether the current date is circled in the calendar or not. An event is also provided to allow you to control which dates are shown in bold.

TwwDBEdit. The *TwwDBEdit* control looks like an ordinary edit box, but can be used with or without a dataset. Its most valuable single feature is support for InfoPower picture masks. Picture masks are supported by all of the InfoPower data-aware controls and are similar in concept to Delphi's edit masks. InfoPower's picture masks let you control the format of text that is entered in a field. For example, you can define a mask that allows either a US five- or nine- digit ZIP code or a Canadian postal code



CheckBook Demo

Paul Coffey
2222 E.55th Street
San Jose, CA 93116

1001

DATE 8/2/1992

PAY TO THE ORDER OF John A. Williams $22.55

Twenty Two Dollars and Fifty Five Cents DOLLARS

FOR: Stereo Cables
John offered me his

:145681 :146 :1001

Signature

Figure 3: A form using transparency and custom framing.

to be entered. You can also create a mask that will automatically capitalize the first letter of each word. Picture masks also provide automatic fill-in of fields. For example, the mask {Red, Green, Blue} will automatically fill in the entire word as soon as the first letter is typed. If the wwDBEdit is bound to a date field and you set the *AutoFillDate* property to True, the user can enter the current date by pressing the spacebar. You can also enable word wrap to display text on multiple lines.

TwwDBLookupCombo. The *TwwDBLookupCombo* box not only allows you to display a list of choices drawn from another dataset, it also allows you to display multiple fields from the lookup table in the drop-down list with or without column headings, with or without column dividers, and with or without row dividers. You can enable Quicken-style incremental searching and the sort order of the drop-down list. You can also use this control without binding it to a dataset, which is ideal for letting users choose values from a lookup table in a dialog box.

TwwDBLookupComboDlg. The *TwwDBLookupComboDlg* looks like the *TwwDBLookupCombo* component, but when its button is clicked, a dialog box appears that contains an edit box, a grid, and a combo box. Typing in the edit box causes an incremental search of the grid to occur. The combo box lets users choose which field to search on. This component is ideal for searching through large datasets.

TwwDBNavigator. In addition to the buttons found in Delphi's DBNavigator, the *TwwDBNavigator* provides next-page and prior-page buttons for moving through a grid a screen full of records at a time. It also includes buttons to set a bookmark, go to a bookmark, display a *TwwDBRecordViewDialog*, *TwwFilterDialog*, *TwwSearchDialog*, or *TwwLocateDialog*. You can also add your own buttons to the Navigator to provide custom functions. The Navigator can be displayed either horizontally or vertically and supports multiple rows of buttons with custom bitmaps. You can also use action lists to provide custom functionality for the Navigator's buttons.

TwwFilterDialog. InfoPower provides a complete suite of controls for finding data lead by *TwwFilterDialog*. The Filter Dialog component works with tables, queries, and even *TClientDataSet* to let users filter their view of data. When filtering a query, you also have the option of letting the database server perform the filter instead of fetching all of the data and performing the filter on the user's workstation. The Filter Dialog has undergone major internal changes for this release to improve its performance. The filter can encompass any number of columns and supports matching on single values or a range of values. Filters on columns can be logically connected using AND or OR and when searching for a single value in a column, you can specify an exact match, starts with, or search for the value anywhere in the columns text. You can also control whether the match is case-sensitive or not, and you can give the user the ability to logically invert the search and see all the records that don't match the specified criteria. The Filter Dialog can filter the data using the *OnFilterRecord* event of the dataset component, the dataset component's *Filter* property, or by building the WHERE clause of a query. By setting a couple of properties, you can ensure that you will get the best possible performance with the back-end database you're using.

***TwwLocateDialog*, *TwwSearchDialog*, and *TwwIncrementalSearch*.** Other search components include *TwwLocateDialog*, *TwwSearchDialog*, and *TwwIncrementalSearch*. *TwwLocateDialog* lets a user easily search for a value in a column using "exact match," "starts with," or "is contained in" options. The user can also control the case sensitivity of the search, and use wild cards in the search string. Find First and Find Next buttons let a user easily step through all the records that match the search criteria. *TwwIncrementalSearch* looks like an edit box, but provides incremental searching on a single field in a dataset. As the user types each succeeding letter, the dataset is dynamically repositioned to the first record that matches the characters typed so far. *TwwSearchDialog* also provides incremental searching, but in a dialog box that displays the dataset being searched in a grid. The user can choose which field to search, and the dialog box can contain a user-defined button that lets you add custom features.

***TwwRecordViewDialog* and *TwwRecordViewPanel*.** Another unique InfoPower component is the *TwwRecordViewDialog* and its cousin the *TwwRecordViewPanel*. The biggest problem with grids is having to scroll horizontally to see all the fields in the current record. The Record View Dialog lets you pop up a dialog box that will show all the fields from the current record in any dataset in a Delphi form that is created automatically, on the-fly when the dialog box is displayed. The Record View dialog box not only creates the form on-the-fly, but it picks the right edit control based on each field's data type. By setting a property, you can control whether the form will have a horizontal or vertical layout. The Record View Panel provides the same functionality in a panel that you can drop on your own custom form.

INFORMANT FACT FILE

InfoPower 2000 continues InfoPower's reputation as the must-have add-in for Delphi. The InfoPower components make any database application easier to develop and more powerful, whether it uses a local table, is a client/server application, or is a multi-tier application. There's no other tool set I've seen that allows you to give your users more power with less effort.

InfoPower 2000 supports Delphi 4 and Delphi 5. The Professional version includes C++Builder 4 support and complete source code for all of the InfoPower components.

Woll2Woll Software
2217 Rhone Drive
Livermore, CA 94550

Phone: (800) WOL2WOL
Web Site: http://www.woll2woll.com
Price: Standard, US$199; Professional, US$299.

Documentation

InfoPower has always been known for its excellent documentation, and version 2000 continues that tradition. In addition to complete online Help, InfoPower 2000 comes with a 294-page spiral bound manual that lays flat on your desk. All the properties, methods, and events are clearly described and code samples are provided where necessary to show you how to take maximum advantage of InfoPower's features.

Conclusion

InfoPower 2000 continues InfoPower's reputation as the must-have add-in for Delphi. The InfoPower components make any database application easier to develop and more powerful, whether it uses a local table, is a client/server application, or is a multi-tier application. There's no other tool set I've seen that allows you to give your users more power with less effort. ∆

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 and over 60 articles, and is 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 Paradox and Delphi programming classes across the country and overseas. He was an instructor on the 1995, 1996, and 1997 Borland/Softbite Delphi World Tours. He can be reached at bill@dbginc.com.

Table of Contents

Mastering Delphi 5

Marco Cantù, co-winner of the 1999 Spirit of Delphi Award, is the author of one of the most popular and highly regarded of all Delphi book series, *Mastering Delphi*. When I reviewed *Mastering Delphi 3* [SYBEX, 1997] in the December 1997 issue of *Delphi Informant Magazine*, I mentioned that Marco Cantù's *Mastering Delphi* [SYBEX, 1995] was one of the first Delphi books I purchased. I continue to return to this series as a trusted reference.

Like its predecessors, *Mastering Delphi 5* covers the Delphi landscape remarkably well. Its comprehensiveness and attention to detail set it apart from many other general introductory texts. The newest edition continues this excellent tradition, and provides valuable information on the new features in Delphi 5. First we'll take a detailed look at the general content and then explore some of the topics covering the new Delphi 5 features.

As in earlier editions, *Mastering Delphi 5* is organized into five large sections. The first part begins with a tour of Delphi's IDE, including such new Delphi 5 tools as To-Do lists, keyboard mapping, and Object Inspector enhancements. Cantù shows how to use the Environment Options dialog box to control new and old features. The next two chapters provide a thorough introduction to Object Pascal. The second of these discusses advanced topics, as well as some of the more recent additions, such as interfaces. Some topics covered in earlier editions (procedural types) have been cut while others (method pointers) remain. The detailed discussion of the various types of strings in Delphi is no longer as relevant, and has been cut. The final chapter in the introductory section deals with the Visual Component Library (VCL).

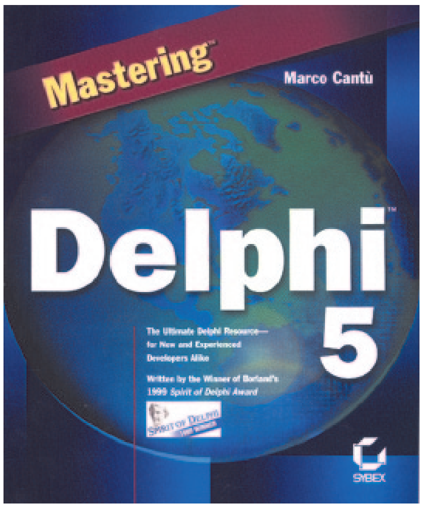
The second part is devoted to using components, beginning with more advanced aspects of working with standard components and issues related to applications, forms, and the user interface. It does an excellent job of covering basic UI issues, such as controlling program behavior with menus, the visual appearance of the various components, dialog boxes, MDI applications, etc.

Part Three is devoted to writing database applications. It is very close to the previous edition with one important change: a new chapter on ADO. This is one of the clearest introductions to Delphi database programming I've seen. It begins with basic concepts, explains the essential components, and provides a wealth of examples demonstrating many common tasks.

Part Four, "Components and Libraries," is reorganized considerably with fewer chapters than *Mastering Delphi 4* [SYBEX, 1998]. In the first chapter, "Creating Components," Cantù includes material previously included in a different chapter. In addition to the basics, you learn about writing component editors, property editors, and experts/wizards. There are also chapters on DLLs, "COM Programming," "Automation and ActiveX" (including OLE), and Internet programming.

The final section of the book is called "Real World Techniques." The first chapter, entitled "Multitasking, Multithreading, and Synchronization," is an excellent exposition of these important topics. Cantù also provides an excellent introduction to Delphi's integrated debugger, shares some debugging techniques and tips, and explains how to handle memory properly. There is a "grab bag" chapter called "More Delphi Techniques," which deals with a plethora of topics from using resources and printing, to working with the Windows Clipboard. The final two chapters explore Internet and multi-tier database programming, respectively.

If you own one of the previous editions of *Mastering Delphi*, you're probably interested in the coverage of Delphi 5 topics. I've already mentioned some. Additional highlights include the new Delphi 5 container classes, which include *TObjectList*, and a discussion of frames. There's a chapter on ADO and Delphi's components that support it. Included is a discussion of moving from Paradox to ADO. There's also an introduction to Delphi 5's new Integrated Translation Environment.


Mastering
Marco Cantù
Delphi 5
The Ultimate Delphi Resource—for New and Experienced Developers Alike
Written by the Winner of Borland's 1999 Spirit of Delphi Award
SYBEX

My opinion of Marco Cantù's excellent book has not changed with this latest edition; it's been strengthened. Admittedly, I have only scratched the surface, but hopefully I've provided sufficient information for you to decide whether this book is right for you. As with its predecessors, it's packed with useful tips, excellent information about new Delphi 5 features, and many practical programming examples. There is no CD-ROM accompanying this book. Instead, you can download all of the code from Cantù's Web site (http://www.marcocantu.com). Further, you can view a complete list of the Delphi 5 topics covered, considerably more than what I have listed. This book is appropriate for all levels of Delphi programmers, but especially for someone who is just getting started with Delphi 5. I recommend it highly.

— Alan C. Moore, Ph.D.

Mastering Delphi 5 by Marco Cantù, SYBEX, 1151 Marina Village Parkway, Alameda, CA 94501, (510) 523-8233, http://www.sybex.com.

ISBN: 0-7821-2565-4
Price: US$49.99 (1,085 pages)

The Future of Computing: Delphi and Linux

In the past year, there has been a great deal of interest in developing a Delphi-like tool to produce Linux applications. This month, I will discuss two contrasting approaches. The first involves an open source project called Megido. The second is Inprise's decision to develop a version of Delphi for Linux. Because most readers of this magazine are Windows developers, I'll begin with a brief overview of Linux.

The World of Linux. Linux is based on UNIX, an operating system developed by AT&T in the late sixties. Like DOS, UNIX was originally a text-based system. The UNIX kernel was originally written in assembly language and later rewritten in C in the early seventies. Because of legal constraints, AT&T was unable to market UNIX. However, it did make this promising operating system available to universities. During the seventies, UNIX continued to develop at various universities, particularly the University of California at Berkeley.

An important tradition began, one that continues today in Linux: the tradition of users helping users. When Sun Microsystems decided to use UNIX on workstations in the eighties, a new trend began. Today, as most readers are aware, UNIX is one of the most popular operating systems for large servers. But what about Linux?

Linux was first developed by Linus Torvalds as a substitute for UNIX. Although it shares many of the features of UNIX, and is largely compatible with its ancestor, there is one important difference. Linux was non-commercial; it was released on the Internet under the GNU general public license. Since then, a large community of developers and users has emerged, a community that works together to develop Linux.

This operating system has many attractive features:
- Memory protection. Each process runs in its own virtual memory space; if one program or process crashes, it won't leave the entire system in an unstable state (as can happen so easily under "other" operating systems).
- Built-in support for multi-users and multitasking.
- Excellent TCP/IP support that is superior to other operating systems.
- Many development tools are available, with more on the way. Many are free.
- Ability to switch from one implementation to another, because Linux is non-proprietary in nature.
- A graphical user interface (GUI) called X-Windows.

This last point may be of particular interest to us because we use Delphi to develop GUI programs for Windows. Just as in Windows, most of the development in Linux is done in C/C++, but not all of it. While working on this column, I did an Internet search (on Northern Lights, my search engine of choice) using just two keywords: Linux and Pascal. I ended up with over 37,000 references! You'll be delighted, and maybe a bit surprised, to learn that one of the folders Northern Lights created to narrow the search was called "Delphi." Many of these links take you to various free Pascal compilers that support various operating systems, including Linux. I will write more about this in future columns. One of the outgrowths of these free Pascal initiatives is a project called Megido, one that appears to be in the early stages of development.

The Megido Project. I first learned about Megido from the following succinct post to the Delphi Advocacy List (TGAD): "These folks are working on building a Delphi/Clone RAD tool for Linux. Looks interesting." I immediately checked out the site at http://www.megido.com (completing this article, I went back to the site and found it under construction). The purpose of that project is stated as: "Do you need a powerful, multi-platform, GPLed, Linux-oriented visual development tool? That's exactly what we are working on!" They list these goals:
- Megido aims to create free Pascal-based RAD tools for the Linux community.
- Megido shares the visual development tool simplicity of programming in all the Linux sophistication.
- Megido is being distributed under GPL/L-GPL, supporting free software movement.
- Megido is using Free Pascal Compiler (GPL), the best Object Pascal compiler, also Delphi compatible.
- Megido is being developed by programmers all around the world who want to contribute to Linux and to break the M$ empire.

Clearly there is no love affair between these folks and Microsoft. In fact, there is considerable discussion on some of the Linux sites about a coming war between Windows and Linux for dominance of the computing market. Prophecy or foolishness? Only time will tell. But there are some interesting indicators that give some credence to the theory that Linux will at least give Windows some competition in the next few years. Corel Corp. has released a version of WordPerfect for Linux. This, in my view, is significant. Will there be a version of Word for Linux? I don't think so. But there will be a version of Delphi.

Delphi for Linux. One of the early indications that Inprise was giving serious consideration to developing a version of Delphi for Linux was a job notice that a colleague in Project JEDI found on the Inprise site: "Senior engineering position responsible for the research and development of major subsystems of Delphi for Linux. Work with the entire team to create Delphi for Linux." Of course there was a significant buzz on this topic at last summer's conference in Philadelphia. Finally on Sept. 28, 1999, the official announcement came that Inprise was "developing a high-performance Linux application development environment" to support C, C++, and Delphi development. They code-named the project "Kylix," and indicated it would be ready for release this year. Further, they promised it would be "one of the first high-performance Rapid Application Development (RAD) tools for the Linux platform." This is something I am really looking forward to.

Admittedly, I have only been able to scratch the surface in this piece. The purpose of this article was to examine certain trends and provide important background information. Next month, I will share with you several books on various aspects of Linux, so that you can learn more about Linux as we all eagerly await Delphi for Linux.

— 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 10 years. He has 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.

table of CONTENTS