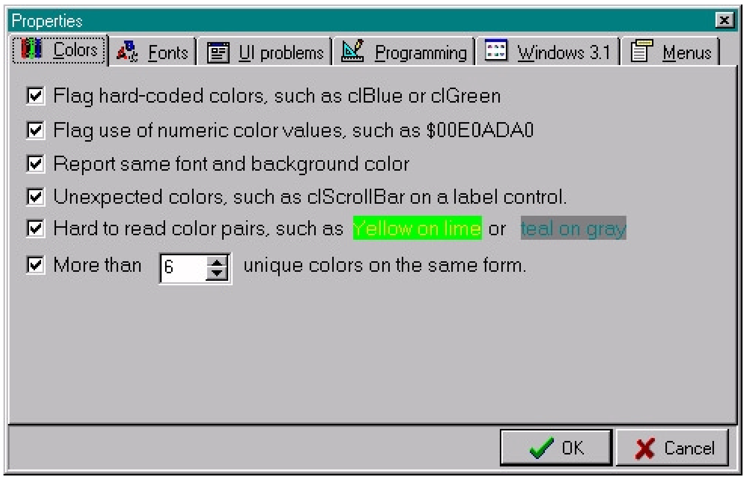
July 2000, Volume 6 Number 7

Cover Art By: *Arthur Dugoni*

Joseph D. Booth Consulting Releases JBC UI/Scan for Delphi



☑ Flag hard-coded colors, such as clBlue or clGreen
☑ Flag use of numeric color values, such as $00E0ADA0
☑ Report same font and background color
☑ Unexpected colors, such as clScrollBar on a label control.
☑ Hard to read color pairs, such as Yellow on lime or teal on gray
☑ More than 6 unique colors on the same form.

Joseph D. Booth Consulting, Inc. released *JBC UI/Scan for Delphi*, a lint checker that allows developers to provide a more professional look and feel to all their application's GUI design.

The developer simply scans an application, and UI/Scan will report tab order problems, menus without code attached, hard-coded colors, fonts that are too small or too large, hard-to-read color combinations, and other user interface concerns.

UI/Scan will assist the developer in putting in the final touches to the visual design of an application. In addition, UI/Scan allows the developer to customize all aspects of a scan. Developers can customize colors, fonts, lists of obsolete controls, etc. There are also filters so UI/Scan can be set to only focus on fixing one problem at a time.

Another feature is the reporting option, which allows the developer to print a report or save it to a file.

Joseph D. Booth Consulting, Inc.
Price: US$149
Fax: (610) 409-8859
Web Site: http://jbooth-consulting.com

Quma Releases QVCS 3.4

Quma Software, Inc. announced the release of ***QVCS 3.4***, a Windows 95/98/NT4/2000 version-control system designed to bring order and control to the application development process.

QVCS automates the tracking of files as they change during the development of a new software application. By requiring programmers to log files in and out, QVCS prevents multiple programmers from unknowingly working on the same file simultaneously.

QVCS makes it easy to control which file revisions are included in which releases, and to recover previous versions of files. QVCS allows you to control executables, documentation, Web pages, and all files associated with a development project.

QVCS stores and retrieves file revisions. You can set up a global journal file to keep track of all changes to files in all your projects. You can insert keywords into source code or binary files for automated documentation and audit trail generation.

At the programmer level, QVCS keeps track of which changes are made to which files, and when. When a programmer checks out a file, QVCS will retrieve the latest version of the file, and record a date/time stamp for that programmer. QVCS ensures that the programmer is authorized to work on the module. It will warn the programmer if a writeable copy of the file already exists, and deny access to the file if another programmer is already working on that file.

When questions arise about program versions, QVCS can be used to compare files to earlier revisions, or compare revisions to other revisions. In an emergency, you can restore a project to an earlier-labeled release.

In addition to using the Windows program, QVCS allows programmers to use batch files to check files in and out, by providing 14 command-line utilities.

QVCS provides a range of reports, including reports of all locked revisions (by individual or team), revisions created after a release, revisions created between any two dates, revisions with comment lines containing any given text string, or any combination of these criteria.

Quma Software, Inc.
Price: US$25; QVCS-Pro, US$40.
E-Mail: info@qumasoft.com
Web Site: http://www.qumasoft.com

O&A Productions Announces oaAgent 1.0

O&A Productions released ***oaAgent 1.0***, a pair of native VCL components that make working with the Microsoft Agent COM server easer. oaAgent brings Microsoft Agent technology to the Delphi and C++Builder IDEs in the form of a non-ActiveX, native VCL wrapper of the Microsoft Agent COM server.

The first component, *ToaAgent*, wraps the Microsoft Agent server COM. Additionally, *ToaAgent* features enhancements not found in the ActiveX control, including persistent voice commands and custom context menus.

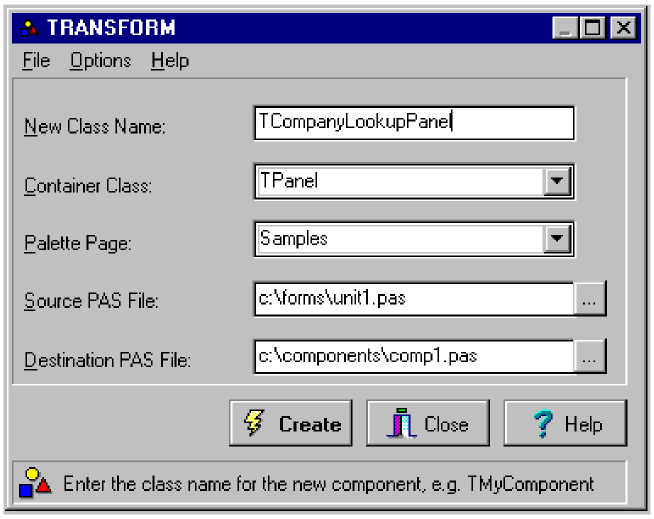
The second component, *ToaAgentScript*, is an extensible scripting language and interpreter component. *ToaAgentScript* manages the interaction of multiple agents and other program elements. Developers can also extend the scripting language to add their own commands.

O&A Productions
Price: US$35
Phone: (858) 618-1904
Web Site: http://www.o2a.com/agent.htm

Objective Software Technology Announces TRANSFORM 5.0.1

Objective Software Technology Pty Ltd. announced *TRANSFORM 5.0.1*, the newest version of the company's Delphi IDE expert that lets you convert entire forms, including form properties and events, into a single component that can be reused inside other forms.

TRANSFORM creates reusable groups of components or "aggregate components," which allow developers to encapsulate the function of a related set of components and selectively publish properties, methods, and events relating to the group.



TRANSFORM
File Options Help

New Class Name: TCompanyLookupPanel
Container Class: TPanel
Palette Page: Samples
Source PAS File: c:\forms\unit1.pas
Destination PAS File: c:\components\comp1.pas

Create Close Help

Enter the class name for the new component, e.g. TMyComponent

The aggregate component can be installed in Delphi's Component palette and reused without having to recreate each individual component.

Aggregate components can be used as business components to encapsulate business logic and user interface controls; to create complex user interface controls, such as toolbars and editors; and as a tool for increasing productivity and object reuse on larger projects.

Objective Software Technology Pty Ltd.
Price: US$125
Phone: +61 8 8357 1805
Web Site: http://www.obsof.com

InstallShield Announces RTPatch for InstallShield Professional

InstallShield Software Corp. announced the availability of *RTPatch for InstallShield Professional*. Developed by Pocket Soft, Inc., and sold through InstallShield and its worldwide network of authorized resellers, RTPatch for InstallShield Professional provides byte-level software patching capabilities specifically for InstallShield Professional 6.x and 5.x setups. End users can now download update "patches" instead of reinstalling the entire program.

RTPatch for InstallShield Professional allows software developers to upgrade a file or a set of files by sending only the changes made since the last release in the form of a simple "patch," which can be easily distributed via an InstallShield Professional setup package. RTPatch typically reduces the actual size of the software update by over 90 percent and provides developers currently using InstallShield Professional with a bandwidth-sensitive solution that enhances the delivery of software updates over the Internet and other mediums.

The GUI-driven Patch Build Wizard walks developers through a simple, four-step process to create a patch. Behind the Wizard, RTPatch's "byte-level differencing" technology (patent pending) compares updated and original software versions to include only the necessary data in the patch.

Once the patch file is created, the developer includes the patch in an InstallShield Professional installation package and distributes it to end users for installation via the Internet or intranet. The end user simply runs the new installation package and the patch is automatically applied.

InstallShield Software Corp.
Price: US$495
Phone: (800) 374-4353
Web Site: http://www.installshield.com

DeVries Data Systems Announces Release of OfficePartner 1.5

DeVries Data Systems, Inc. announced the release of *OfficePartner 1.5*, the latest version of its suite of software components designed to integrate the Microsoft Office suite with Borland software development tools.

OfficePartner's expanded functionality will enable developers to produce results without a great deal of Automation expertise. With new functionality added to all main components, automating Office applications from Delphi and C++Builder is easier and more stable.

New features include expanded and enhanced documentation to provide how-to and reference information; mail merge capability in Word components that supports VCL Datasets and ODBC; an expanded connection framework that allows OfficePartner components to attach to running instances of Office applications and OLE containers; find-and-replace functionality included in Word components; print methods for Word documents and Excel workbooks; a new element for accessing subfolders in Outlook 98/2000; and the ability to run in hundreds of development tools/-platform scenarios, including the latest versions of Delphi and C++Builder.

DeVries Data Systems, Inc.
Price: US$399 per user; discounts available for multi-license packages.
Phone: (888) 866-8031
Web Site: http://www.dvdata.com/dvnew/pages/index.asp

Lingscape Announces MultLang Suite 3

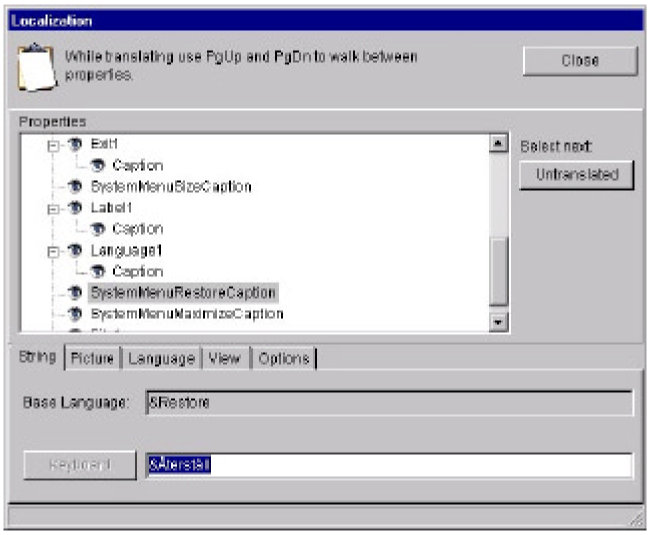
Lingscape Ltd. announced *MultLang Suite 3*, a new version of the company's globalization tool for Delphi and C++Builder. MultLang Suite targets general, intermediate, and advanced developers involved with international projects with full quality control.

MultLang Suite 3 helps reduce internationalization and localization costs using wizards and machine translation. It is based on the Unicode standard, and, together with a conversion engine, provides target support for languages such as Japanese, Chinese, Arabic, Hebrew, Hungarian, Russian, and all European languages.

The integration with the IDE and the compiler helps produce thin localized EXE, DLL, and ActiveX, or optionally link multi-language support into the same application. With MultLang, it's possible to compile Far East, Middle East, and European languages together on the same platform.

Unique features include one code base for all languages, machine translation, quality assurance guidelines, form validation routines to detect form defects, leverage feature for previous work, migration utility to dynamically retrieve localized editions, team development, context-sensitive dictionaries, and more.

Lingscape Ltd.
Price: US$998; printed manual/CD/-postage, US$39.
E-Mail: info@lingscape.com
Web Site: http://www.lingscape.com



While translating use PgUp and PgDn to walk between properties.

Base Language: &Restore

&Aterställ

EliteSys Announces SuperBot 2.2

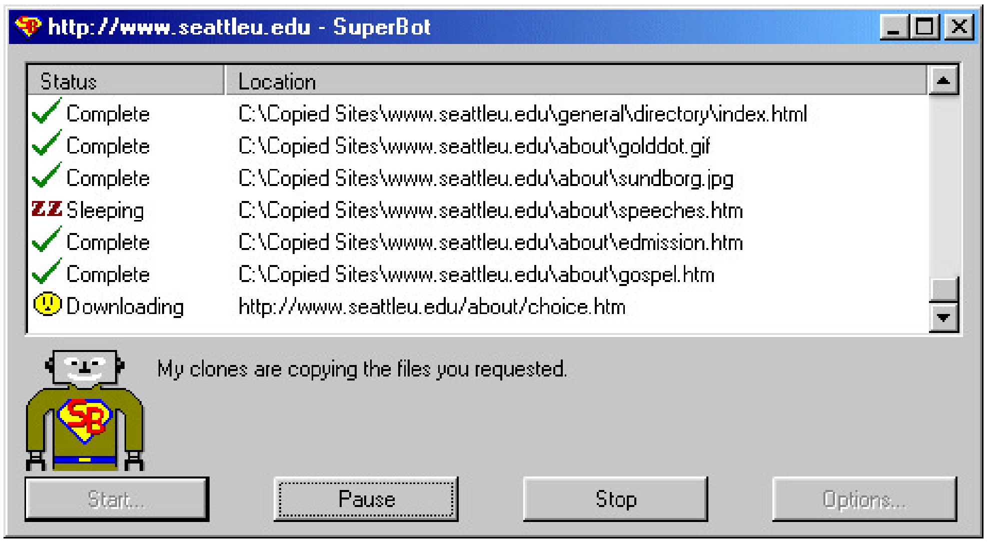
EliteSys announced the release of *SuperBot 2.2*, an automated download utility for Windows 95/98/NT4 that can copy entire Internet sites with one click. Thanks to SuperBot's HTML rewriting technology, copied sites look and act just like their online counterparts.

Once SuperBot has downloaded a Web site, it can be viewed in any Web browser, at high speed, and without an Internet connection. Modem users are therefore able to surf the Net without missing calls. A copied site can be transferred directly to any other storage medium for archival or distribution purposes.

SuperBot can be configured to filter downloads by location, date, file type, link type, depth, and file count. Once a copy operation has begun, it can be paused, stopped, and restarted. SuperBot utilizes HTTP/1.1 "Smart Restart" technology to resume any partially completed file transfers.

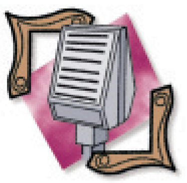
With the "Monitor clipboard" feature, you can download a Web site by merely copying its address and pressing Enter. SuperBot can automatically launch a copied site in your browser, as well as record its location in a log file.

EliteSys
Price: US$24.95
E-Mail: elitesys@iname.com
Web Site: http://web.idirect.com/~elitesys/superbot



Status	Location
✓ Complete	C:\Copied Sites\www.seattleu.edu\general\directory\index.html
✓ Complete	C:\Copied Sites\www.seattleu.edu\about\golddot.gif
✓ Complete	C:\Copied Sites\www.seattleu.edu\about\sundborg.jpg
ZZ Sleeping	C:\Copied Sites\www.seattleu.edu\about\speeches.htm
✓ Complete	C:\Copied Sites\www.seattleu.edu\about\edmission.htm
✓ Complete	C:\Copied Sites\www.seattleu.edu\about\gospel.htm
Downloading	http://www.seattleu.edu/about/choice.htm

My clones are copying the files you requested.



TurboPower Announces Support for C++Builder 5

Colorado Springs, CO — TurboPower Software Co., developer of tools and libraries for professional Delphi and C++Builder programmers, announced full and free support for Inprise Borland C++Builder 5. Compatibility updates will be available for all current version TurboPower products. The free updates will be available as no-charge downloads from the company's Free Update Center at http://www.turbopower.com/updates/. Updates will also be available on CD-ROM for a nominal charge.

In addition, customers can subscribe to free TurboPower electronic newsletters to receive instant e-mail notification when patches for the products they use are available for download. The address for e-newsletter subscriptions is http://www.turbopower.com/tpslive.

To learn more, visit http://www.turbopower.com, or call (800) 333-4160.

Inprise/Borland's Kylix Project Builds Third-party Network for Linux

Scotts Valley, CA — Inprise/-Borland hosted more than 200 third-party authors, consultants, trainers, and tool and component vendors for the first in a series of worldwide events designed to prepare third-party products and services for Kylix.

Also participating were Linux distributors, including MandrakeSoft, TurboLinux, Corel, Caldera, and SuSE. Announced in September 1999, Project Kylix will be a Linux rapid application development environment that will support Delphi, C, and C++.

The Kylix project is planned to be the first high-performance rapid application development tool for the Linux platform. The Kylix project is a component-based development environment for two-way visual development of graphical user interface, Internet, database, and server applications. Kylix will be powered by a new native Delphi/C/C++ compiler for Linux and will implement a native Linux version of the Borland VCL (Visual Component Library) architecture. The Borland VCL for Linux is designed to radically speed native Linux application development and simplify the porting of Delphi and C++Builder applications between Windows and Linux.

Inprise/Borland Opens Public Field Test of InterBase 6.0

Scotts Valley, CA — Inprise/-Borland announced a public field test of InterBase 6.0, its cross-platform relational database, for the Linux, Windows, and Solaris operating systems. A beta version of the database — which will be open-sourced with multiple platform support in mid-2000 — is now available as a free download from the InterBase Web site at http://www.interbase.com/open/downloads/.

Inprise/Borland invites interested parties to freely download and test this latest version of InterBase for Linux, Windows, and Solaris. Any feedback on this new version can be sent to ib_support@inprise.com. Alternatively, users are encouraged to participate in the field test newsgroups at http://www.interbase.com/open/community/60beta_newsgroups.html.

Field test versions of InterBase 6.0 for Windows, Solaris, and other operating systems will be made publicly available in the near future.

To learn more, visit Inprise/Borland at http://www.borland.com, the community site at http://community.borland.com, or call the company at (800) 632-2864.

Inprise/Borland Java Development Tools Win Multiple Awards

Scotts Valley, CA — Inprise/-Borland announced that two of its popular Java development tools, JBuilder and JBuilder JIT, have won multiple awards over the past weeks. JBuilder, a rapid application development (RAD) environment for Java development, won both the *Software Development Magazine* Jolt Product Excellence Award in the Language and Development Environments category, and the 1999 *JavaWorld* Readers' Choice Award for best IDE (integrated development environment). The JBuilder JIT, Borland's Just-In-Time compiler, won the *JavaWorld* Readers' Choice Award for best compiler.

Inprise/Borland and Hitachi Strengthen Co-development Relationship

Scotts Valley, CA — Inprise/-Borland Corp. and Hitachi, Ltd. announced they have signed a worldwide amendment to an existing licensing agreement for CORBA and Java development technology.

Under the terms of the new agreement, the research and development laboratories of the two organizations will begin to share their own innovations to Inprise/Borland's VisiBroker product line, including source code for functional innovation, technical enhancement, and quality improvement.

Amendments to the current agreement extends the relationship between the two companies, reaching back to 1995, for co-developing CORBA-based products and will broaden the commitment of the two organizations to share enterprise middleware technical innovations.

As provided in a previous agreement, Hitachi will continue to license Inprise/Borland's VisiBroker and Inprise's Java development technology for inclusion in Hitachi's CORBA technology products, such as TPBroker, Hitachi's CORBA object transaction service product, and the Cosminexus Application Server. (Cosminexus Application Server is currently marketed only in Japan.)

Financial terms of the agreement were not disclosed. For more information on Hitachi, visit http://www.hitachi.co.jp.

By Dr Mark Brittingham

Real-world Web Apps

Building Session-aware ISAPI DLLs

If you've been working in Delphi for a while, you know it's the most productive Windows development tool available. What you may not realize is that it's also an excellent tool for developing Web applications. In this article, we'll cover how to build a session-aware ISAPI DLL that can be used in a real-world application.

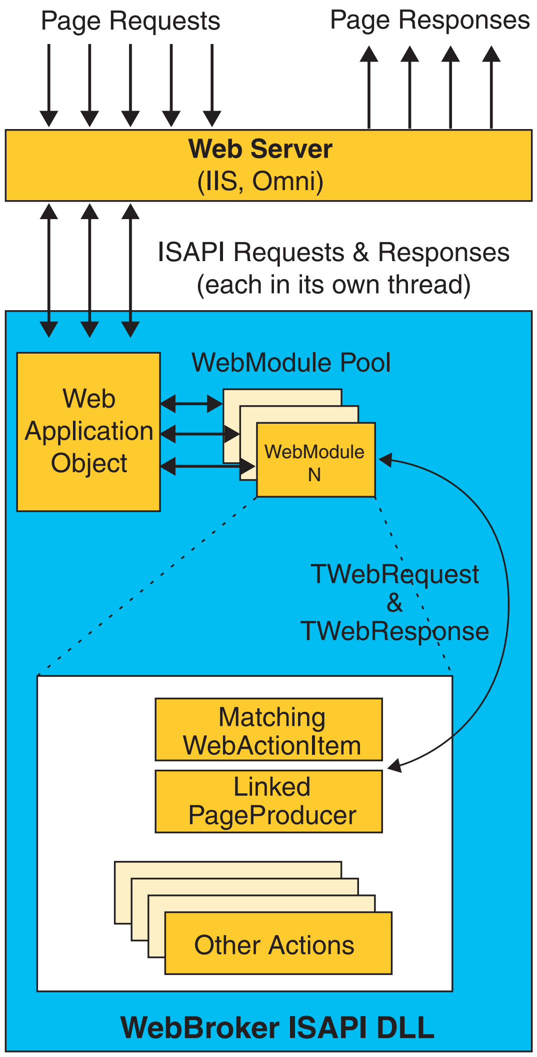


Figure 1: Delphi's ISAPI architecture.

If the rise of Internet technologies has you worried that you'll have to leave the world's best development environment behind, fear no more. Delphi's WebBroker technologies give you an easy way to build fast, scalable Web applications.

Web applications are defined by a client-server model, where the client is a Web browser that interacts with the Web server by sending page requests and form results and receiving HTML in response. In a very important sense, your job as a Web developer is to "program" the user's browser to behave the way you want it to by sending it the appropriate HTML and JavaScript code.

Getting Started

Before starting, make sure your environment has all of the tools needed for building your site. You'll need a Web server, a Web browser, an HTML editor, and either Delphi Enterprise or Delphi Professional with the WebBroker libraries (also available at extra charge from Inprise). We assume that you'll be developing an ISAPI DLL because of the very significant boost in performance these Web applications provide. Not only do they avoid the overhead of loading a CGI executable for every page request, they permit you to create an extremely fast state management system in memory, rather than relying on database access to maintain state.

For a Web server, you may choose to use Microsoft's Personal Web Server (PWS) or Internet Information Server (IIS) because these come free with Windows 98, NT, and 2000. However, I recommend that you download and install the OmniHTTPd server from Omnicron. It's available at their Web site: http://www.omnicron.ab.ca/httpd. Omni is far easier to use as a debugging host in Delphi than the Microsoft Web servers, and is free for local and development use. When it comes time for deployment, you can either substitute IIS, or license the commercial Omni distribution. Because both support the same ISAPI extension standards, they're interchangeable.

In practical terms, your work will proceed by creating your HTML pages, building and compiling your ISAPI DLL, and refreshing your browser to review the results. Because your browser and server are on the same machine, your Web URLs will all start with http://localhost/. Behind the scenes, the browser will contact the server, and the server will load your DLL and request the HTML to be sent on to the browser.

Note that you won't be able to compile your ISAPI DLL while the server is running, because Delphi won't be permitted to overwrite the old DLL being run by the server. Thus, the server will have to be shut down each time you're finished testing and started up again before requesting another page. To shut down Omni, just use its icon in the system tray. Use the Internet Services Manager to stop PWS or IIS.

Life will be considerably easier if you configure Delphi to run the server as the **Host Application** under the **Run | Run Parameters** menu. For the Omni Web server, you need only to place the location of the "ohttpd.exe" executable in the host field. Configuring the Microsoft Web servers is far more complex, although configuration for IIS 4.0 is well documented at the "D files" Web site at http://www.fulgan.com/delphi/index.asp.

If this will be your first ISAPI application, then you need to know how to tell Delphi to create an ISAPI DLL. Do this by using the **File | New** menu. On the New tab in the New Items dialog box, select Web Server Application. In the dialog box, select the **ISAPI/NSAPI Dynamic Link Library** radio button, and press **OK**. You're in business!

Delphi's ISAPI Architecture

Writing raw ISAPI DLLs isn't rocket science, but it still involves quite a bit of specialized knowledge and a knack for thread-safe coding. Fortunately, Delphi's WebBroker libraries make it much less tedious, and a lot more fun. Figure 1 captures the essential components of a WebBroker application. As you can see, everything is driven by requests from the server.

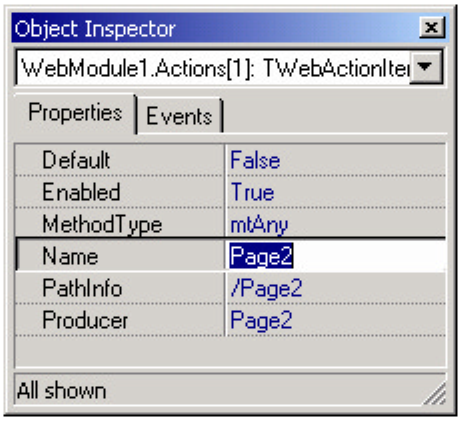
When the Web server receives a request, it bundles the data related to the request and sends it in a new thread to the ISAPI DLL. The *TWebApplication* object in the DLL, in turn, uses an existing WebModule, or, if necessary, creates a new WebModule instance to service the request. This WebModule executes within the thread context passed by the server. This means you can use variables defined within the *TWebModule* class and be assured they aren't shared with other threads. However, global variables will be shared across threads, so you should avoid them, or be certain your access is thread-safe.

If you need data access, you'll be happy to know that the BDE and ADO datasets can be placed on the WebModule and used in a thread-safe manner. The only constraint is that, for the BDE, a *TSession* object must be placed on the WebModule, and its *AutoSessionName* property must be set to True.

A URL that calls an ISAPI DLL should name an action to be executed. For example, a URL like:

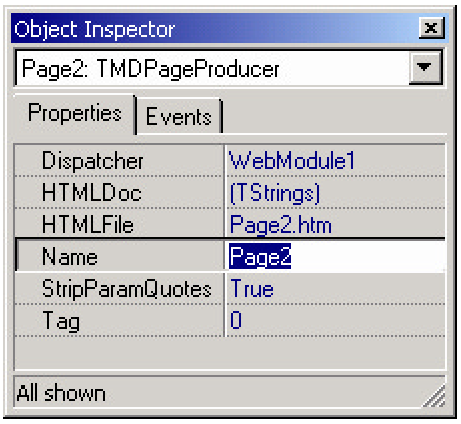
```
www.Mysite.com/ISAPI/MyIsapi.dll/Signin
```

will call MyIsapi.dll and hand it the Signin action. When the WebModule receives the request, it attempts to locate a *TWebAction* whose *PathInfo* matches the action passed in the URL (see Figure 2). You should generally create one action item with a *Default* property set to True to handle the case where no other action item handles a request made to the DLL.



Default	False
Enabled	True
MethodType	mtAny
Name	Page2
PathInfo	/Page2
Producer	Page2

Figure 2: A Web action item in the Object Inspector.



Dispatcher	WebModule1
HTMLDoc	(TStrings)
HTMLFile	Page2.htm
Name	Page2
StripParamQuotes	True
Tag	0

Figure 3: A PageProducer in the Object Inspector.

Note that, in addition to the *PathInfo* property, this action item has a producer named *Page2*. In my development, I give the *PathInfo*, the action, and the producer the same name to make it clear they all work together. (Yes, you can give an action and a producer the same name.)

There are two ways in which you can respond to a request arriving at your DLL: with a producer, or via the action item's *OnAction* event. If you respond to the *OnAction* event, you'll have to generate your HTML and pass it back by assigning the *Response.Content* field:

```
Response.Content := SomeHTMLGenFunction;
```

If you're using a PageProducer, you'll generally assign a file to the PageProducer's *HTMLFile* property (see Figure 3). This will cause the file's contents to automatically be sent as a response.

If you wish, you can keep your entire HTML document in a PageProducer's *HTMLDoc* property. The advantage of this is that you won't have to distribute HTML pages with your DLL. The disadvantage is that your site is much more difficult to update: Every update will require a re-compile, and your work will slow down every time you need to cut-and-paste your HTML between Delphi and your HTML editor. Personally, I never use the *HTMLDoc* property.

Of course, if your DLL did nothing more than pull HTML files from disk using PageProducers, then there would be no sense in creating the DLL. The PageProducer family (*TPageProducer*, *TQueryTableProducer*, *TDataSetTableProducer*, and

TDataSetPageProducer) is powerful because of its *OnHTMLTag* event. The *OnHTMLTag* procedure for a PageProducer is called during the parsing of the *HTMLFile* or *HTMLDoc* streams. Every time a tag (marked with <#> delimiters) is encountered, this function receives a call. The call contains the tag, any arguments to the tag, and a **var** parameter named *ReplaceText* that you can fill with the output you want to appear in place of the tag.

For example, if your PageProducer is loading a file named DateTime.htm, and it encounters the tag <#Date>, then the *OnHTMLTag* function will be called with the <Date> tag. After matching on the appropriate tag in this function, you could fill the *ReplaceText* parameter with something like "March 15, 2000." When the HTML is sent to the browser, this date will appear in place of the tag.

To close out the topic of basic ISAPI development, note that one of the best things that Delphi does for you is take all of the form, URL, and cookie arguments submitted by a visitor and package them neatly in the *ContentFields*, *QueryFields*, and *CookieFields* string lists. For example, if you want to know the value a user recorded in the "Name" field of a form, you need only look at the result contained in:

```
Request.ContentFields.Values['Name']
```

To become a good ISAPI developer, working with string lists must become second nature!

Dynamic Web Pages and Sessions

After the elation of building your first ISAPI DLL, you'll probably come to the realization that you're nowhere close to a real Web application. This is because the Web is a completely stateless environment. In standard Windows development, you can pop up a dialog box, have the user fill in some answers, and return to the main form without any worry that a different user has requested each of these actions. This isn't true on the Web! Each request your DLL receives is logically independent of every other request.

That means that, although your simple ISAPI DLL can now produce a Web page that includes dynamically-generated information, it will either generate the same information for every user, or will use only the immediately preceding page to generate the response, e.g. processing a form page.

To get around this limitation, we need to implement some mechanism for state maintenance. A state mechanism permits you to store information about a visitor (their "state") so that each page request has access to any information the visitor entered on any earlier page. A prime example of the utility of state management is an e-commerce application that must remember the products a visitor has selected, their name and address, and their payment information, even though all of this information has been entered on different pages at different times.

Typically, applications also differentiate between a user's session (state information stored in the course of their current visit) and a user's permanent information. Of course, information entered in a session usually becomes part of a user's permanent information. However, sessions are usually set up to expire after a relatively brief period of inactivity (20-30 minutes).

The Art of Maintaining State

There are only three mechanisms for identifying and tracking a visitor as they progress from page to page in your site: forms, cookies, or fat URLs. In all three cases, you might consider storing the user's entire state in the cookie/URL/form. However, unless you're implementing a very small site, it's far more appropriate to include a numerical ID in the cookie/URL/form that serves as the key to finding that information on the server.

```
procedure TWebModule1.WebModuleBeforeDispatch(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
var
  cookieStrings : TStringList;
begin
  SessionID :=
    StrToIntDef(Request.CookieFields.Values['SID'], 0);
  if (SessionID = 0) then  // No sessID passed.
    begin
      SessionID = Random(2000000000);
      cookieStrings := TStringList.Create;
      cookieStrings.Add('SID=' + IntToStr(SessionID));
      Response.SetCookieField(
        cookieStrings, ", ", Now+10, False);
    end;
end;
```

Figure 4: Using a cookie to manage a session ID.

Forms. To maintain state using forms, you would have to place every link on your site in an HTML form, and store the state ID in a hidden field. Then, as the user navigates the site, the ID would be passed in the *Request.ContentFields* variable in each request. This isn't a viable approach, because of the overhead and pain involved in implementing every link as a form.

Cookies. To maintain state in a cookie, you'll generate a unique session or user ID, and store it in a cookie that is placed on the visitor's computer when they first enter the site. On subsequent pages, this cookie is pulled from the page request and used to access the visitor's state information. Figure 4 shows Delphi code to accomplish this. Note that I've placed this code in the *WebModuleBeforeDispatch* method of the WebModule. This is because *BeforeDispatch* is called at the beginning of every page request, and is thus an ideal place to set up the session for the rest of the request. Note that all functions called in the process of satisfying a request are within a single thread. This means you can assign a variable in one function, and use it in another if the variable is defined in your *WebBroker* class.

The advantage of a cookie-based solution is that you can set the cookie up in your DLL just once, and not have to worry about it again. Also, you can use cookies to store a permanent ID for a visitor so that when he or she returns, you can immediately provide them with personalized information.

The disadvantage of cookies is that some users turn them off. There is a great deal of fear-mongering, and downright ignorance, when it comes to cookies — as well as some legitimate concern — so it's not surprising that some people disable them. If you're using cookies to maintain state, your site simply won't work for people who turn them off.

It's also important to recognize the distinction between using cookies to maintain state, and using them to uniquely identify a user. If you use cookies to maintain state, they should expire soon after their last use. If you use cookies to identify a user, it's very important that he or she have some control over this process. Not all computers are used by a single individual, so storing a cookie on the assumption that the same visitor is using the cookie on each visit could lead to confusion, or worse, to a significant breach of personal privacy.

Fat URLs. Another solution is to embed a session ID in the URLs visitors use as they navigate your site. To do this, you will generate a unique identifier when the first page is requested. On all subsequent pages, any internal links will have an argument in their URL that passes along the ID. For example, an internal link in your raw HTML page might be:

```
<a href='/ISAPI/Demo.dll/page2.htm?SID=<#SessID>'>
```

In your page-handling code, you must ensure that every page handles the <#SessID tag>, substituting the current session ID appropriately. So, to your user, that internal link shown above might be:

```
<a href=' /ISAPI/Demo.dll/page2.htm?SID=374834'>
```

In all page requests after the first, you would pull the session ID from the *Request.QueryFields* StringList whenever you needed to access session information.

As you may already be thinking, the drawback of this approach is that you not only have to include the session ID tag in every internal link, you also have to write code in every action item or PageProducer to substitute in the session ID. Also, every page must be run through your DLL, even if it wouldn't otherwise need dynamic processing. This is because it will now need dynamic processing to substitute in the session ID.

Magic bullet. It's tempting to believe there is some other magic bullet for state management. Indeed, Web tool vendors often advertise "automatic state management," and may even imply that they use some method beyond or outside of this set. However, there isn't any magic in Web development, and, upon closer inspection, every vendor's state management always comes down to one of these methods, or some combination of them.

Even with these tips for keeping a session ID around, you may still be wondering how you can maintain a robust, thread-safe repository for session information. If so, grab a SoBe (or a cup of Joe) and settle in. We're heading for the Delphi Web Application Component power tour.

The MDWeb Components

One of the truly great things about Delphi is that you can simply subclass and override any behavior you don't like in the VCL's components. I truly like the *WebBroker* libraries; they do a lot to make my life easier. However, they really just don't do enough to make Delphi a contender in real-world Web development. And that's where Delphi's strength in component subclassing comes in handy. The MDWeb components, derived from Delphi's PageProducer family, take all of the WebBroker's strengths and add automatic, thread-safe session management. If you follow the discussion of the MDWeb components, you should gain a good idea of how to implement your own session management controls. Of course, you're also welcome to visit the *Delphi Informant Magazine* Web site and download the MDWeb components that accompany this article (see end of article for details).

The heart of the MDWeb components is *TMDSessionMgr*. This class is derived from *TPageProducer*, and has been modified to do two things:

1) Automatically replace "session-level" tags (Session ID, Date) in every file that passes through the system.
2) Provide seamless access to an in-memory session variable store (a B-tree implementation). Note that the session manager doesn't handle any particular page request; it works with all page requests.

You develop with the MDWeb components just as you do with standard WebBroker components. However, you must drop an MDSessionMgr component on your WebModule and make sure it's moved to the first position in the creation order list. This ensures that basic session management capabilities are implemented. Then, instead of using WebBroker PageProducers, you use the corresponding MDWeb components.

The operation of the MDWeb components is identical to the WebBroker components, except that when these components are finished processing their HTML, they call the MDSessionMgr so it can process any session-level tags. These can include any tags you want to support on a global basis: Session ID, Date, Time, random number generation, or even file includes.

If you're using URLs to maintain state, you'll embed the <#SessID> tag in the links that tie your pages together as previously described. When a page is sent to a visitor, its Session ID will be automatically inserted into these links so that subsequent page requests pass the ID along. Although you still have to ensure that all of your page links have the embedded session tag, this automatic tag substitution makes URL-based session management quite a bit more tractable. Note that the MDWeb library also provides a function that can be used in an action item's *OnAction* event to simplify the loading of pages that need no other processing than the substitution of session-level tags.

```
TMDSessionMgr.ContentFrom-File(filename: string): string;
```

If you use this function, you can load an HTML file without having to create a PageProducer. Of course, you may still prefer to use cookies to store/access the session ID to avoid the hassle of embedding the ID in the URL.

While global substitution is an important task for *TMDSessionMgr*, the big advantage in using this class is its management of session data. When a visitor arrives at the site, their first page request won't have an associated session ID in the URL or in a cookie. When the DLL sees this request, it will trigger the creation of a session data object (see Figure 5). When created, this object will automatically generate a random session ID in the range of 2 to two billion. In addition to the session ID, the session data object holds a string list in which all data for a session will be stored.

The session data is stored in a thread-safe, in-memory, B-tree for fast access. The *TMDSessionMgr* object (shown in Listing One, beginning on page 11) hides all access to the B-tree, so you can swap in a different storage mechanism without having to change your project code. For example, if your site is so busy that you need to move to another server

```
procedure TWebModule1.WebModuleBeforeDispatch(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  SessMgr.SessID :=
    StrToIntDef(Request.QueryFields.Values['SID'], 0);
  if (SessMgr.SessID = 0) then  // No session ID passed.
    SessMgr.CreateSessionObject
  else
    if (Not SessMgr.FindSessionObject) then
      Response.SendRedirect('/Timeout.htm');
end;
```

Figure 5: The *OnBeforeDispatch* procedure.

or servers, you could move session data to a shared database, and access it from within your *TMDSessionMgr* object instead of the B-tree.

You may or may not choose to make use of our sample code when starting your session-aware project, but the outline should be clear: A session ID must be generated if a user doesn't already have one, and this ID must index a data structure or database record holding all of a user's session information. If you use a database table to hold session information, you won't have to worry about the thread safety of your data access, but you may have performance issues. If you do store session information in memory, you should gate your data access with critical sections to ensure thread safety.

If you choose to manage sessions by embedding session IDs in the intra-site URLs, some mechanism for simplifying the substitution of actual session IDs should be implemented as well.

Enough Theory!

Now that you know some session management theory, let's see how things work in a sample application. Recall that the first thing a Delphi Web Application does when it receives a request is call the *OnBeforeDispatch* function of the WebModule. This occurs even before an action is selected to handle the request. This is the perfect opportunity to check whether the current visitor has been assigned a session, and to either create or find a session object that can be used in the remainder of this request. Again, refer to Figure 5 to see the code to do this.

In the example in Figure 5, we attempt to pull a session ID from the URL using the *Request.QueryFields* parameter. Thus, if the URL requesting this page is:

www.mysite.com/ISAPI/Demo.dll/Action?SID=1234

then the "SID" index to the *QueryFields* StringList will retrieve the value "1234." If no SID argument was passed, then the result of this retrieval would be 0. Note that we could use cookies instead by simply using `Request.CookieFields.Values['SID']` in the statement. We would also have to make sure the cookie is stored in the call to *CreateSessionObject*.

If the Session ID is 0, then this request comes from a new visitor. In this case, we generate a new session object with the *CreateSessionObject* procedure. Otherwise, we tell the *SessMgr* to get the session object ready for use with the *FindSessionObject* function. In either case, the result is the same: The *SessMgr* object in the current thread will now be able to store or retrieve user values specific to the current visitor. The actual storage or retrieval will occur as this particular request percolates through the Web action items and/or *MDPageProducer* classes. Let's see how this happens.

Assume for a moment that our site needs to assess the current color preference of our visitor. This information will be used later in the visit, and thus must be stored in the current user's session variables for later use. The HTML for the color request form is shown in Figure 6.

```
<FORM METHOD="post"
  Action="/ISAPI/DemoWeb.dll/Page3?SID=<#SessID>">
<TABLE WIDTH="550" CELLPADDING="0" CELLSPACING="0"
  BORDER="0" ALIGN="center">
<TR>
<TD ALIGN="right" WIDTH="50%">
      What is your favorite color?&nbsp;<BR>
</TD>
<TD ALIGN="left">
  <INPUT NAME="COLOR" MAXLENGTH="12" ><BR>
</TD>
</TR>
<TR>
<TD ALIGN="middle" COLSPAN="2">
<BR>
<INPUT TYPE="submit" NAME="Submit" VALUE="  Submit  ">
&nbsp;&nbsp;
<INPUT TYPE="reset" NAME="reset" VALUE="  Reset  ">
</TD>
</TR>
</TABLE>
</FORM>
```

Figure 6: The color preference HTML form.

When the color preferences form is filled out and submitted, the *Page3* action will be requested. Note the inclusion of the *SessionID* in the page request. It's in our response to this new action that we process the information sent from this form. In this case, we store the color preference via a function call in the action item that handles the page request:

```
procedure TWebModule1.WebModule1Page3Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  SessMgr.Values['COLOR'] :=
    Request.ContentFields.Values['COLOR'];
end;
```

In case you're wondering, the actual page request is handled by an *MDPageProducer* class (see Listing Two on page 13). Recall that there are two ways to respond to a page request: via an *OnAction* procedure in the action item, and via a PageProducer (or TableProducer, etc.). As shown here, it's also acceptable to use both. However, a word of warning is in order. You may be tempted to pull some data from your database in the *OnAction* procedure hoping to use it in the PageProducer when responding to tags in the page. This won't work if you attach the PageProducer to the action item, because the WebBroker will always call the PageProducer before the *OnAction* procedure. This seems to be glaringly non-obvious behavior for the WebBroker. Fortunately, a work-around is easy enough: Access the PageProducer manually in the *OnAction* procedure, and don't link the PageProducer and action item. For example, in an *OnAction* procedure, you might write the code shown in Figure 7.

```
procedure TWebModule1.WebModule1SummaryAction(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  SessMgr.Values['MOVIE'] :=
    Request.ContentFields.Values['MOVIE'];
  // Ask the "Summary" PageProducer to get the page.
  Response.Content := Summary.Content;
end;
```

Figure 7: Getting information from a form before generating a response page.

```
procedure TWebModule1.SummaryHTMLTag(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse;
  var Handled: Boolean);
begin
  if (AnsiCompareText(TagString, 'FOOD') = 0) then
    ReplaceText := SessMgr.Values['FOOD']
  else if (AnsiCompareText(TagString, 'COLOR') = 0) then
    ReplaceText := SessMgr.Values['COLOR']
  else if (AnsiCompareText(TagString, 'MOVIE') = 0) then
    ReplaceText := SessMgr.Values['MOVIE'];
end;
```

Figure 8: Using session values.

Note that the `Response.Content` string is where all the HTML to be streamed out to the user is stored. We fill it manually here with the output of the Summary PageProducer. In a database application, you would place your database access code before the call to the PageProducer, if the producer needed data to perform its tag substitutions.

Now that you've seen how session data is stored, it's probably pretty obvious how to access this information. For purposes of illustration, however, see Figure 8, wherein we pull all of the visitor's favorites and use them in tag substitutions.

As you can see, we pull data the same way we set the data. There are also functions for storing and accessing integer values in the MDWeb library.

Conclusion

The problem I sometimes have in explaining session management is that storing and accessing data in the Session Manager seems pretty boring and obvious. It's not! Keep in mind that a busy site might have hundreds or even thousands of visitor sessions running concurrently. The data storage and access for every one of these visitors must be kept straight. The art isn't in storing and retrieving the data; it's in doing so in a manner that ensures that every visitor sees only their own information.

Having a lightweight mechanism for maintaining state is the gateway to all of the more advanced work you'll do on the Web. Do you want to create an e-commerce site? Do you want to provide a highly personalized experience for your visitors? Need to manage logins and secure site access? In all of these cases, session management is the foundation from which you will build. ∆

All source described in this article (including the B-tree code) is available on the Delphi Informant Magazine Complete Works CD located in INFORM\00\JUL\DI200007MB.

Dr Mark Brittingham believes the secret of happiness is to radically change careers every four to five years. He has worked at Bell Laboratories (now Lucent Technologies) in Artificial Intelligence Research, at AT&T in user interface design, and as president of Brittingham Software Design. He created a Windows-based vertical market health and fitness package whose royalties now pay the bills. He is currently doing Web development in Delphi and Cold Fusion as a freelance Internet consultant.

Begin Listing One — *TMDSessionMgr* Object

```
unit MDSessMgr;

interface

uses
  Windows, HTTPApp, Classes, SysUtils, MDBTree;

type
  TMDSessionMgr = class(TPageProducer)
  private
  protected
    function HandleTag(const TagString: string;
      TagParams: TStrings): string; override;
  public
    SessID : Cardinal;
    SessNode : PTNode;
    procedure DoTagEvent(Tag: TTag;
      const TagString: string; TagParams: TStrings;
      var ReplaceText: string); override;
    function ContentFromFile(filename: string): string;
    procedure CreateSessionObject(inUID: Integer = 1);
    function FindSessionObject: Boolean;
    procedure SetValue(const Name, Value: string);
    function GetValue(const Name: string): string;
    procedure SetIntValue(const Name: string;
      const Value: Integer);
    function GetIntValue(const Name: string): Integer;
    property Values[const Name: string]: string
      read GetValue write SetValue;
```

An Extension of Your Own

ISAPI URLs are pretty ugly and hard to remember. A typical URL might look like this:

www.something.com/ISAPI/Your.DLL/Action

However, there is a way to skip the explicit DLL call and simply name the files you want to use in your URLs. All you need to do is make a simple change in your Web server, and a matching change in your Delphi DLL. In the examples that follow, I assume that you'll name your HTML files with a .dwm extension (for Delphi Web Markup) in order to clearly differentiate them from standard .htm files.

To set up .dwm files in Omni, you have to make an addition to the External files listing. To do this, select **Properties | Web Server Globals Settings** and click the External tab. Once there, you'll add a new entry. In the **Virtual** field, place the name of your extension: .dwm. In the **Actual** field, place the path to the DLL you are developing. Next, click on the MIME tab, and add a new MIME type: enter wwwserver/isapi in the **Virtual** field, and .dwm in the **Actual** field. In the External and the MIME tabs, be sure to press the **Add** button after filling in your fields!

In IIS 5.0, you'll right click on the HTTP server and select **Properties**. In the resulting dialog box, select the Home Directory tab and press the **Configuration** button. In the dialog box that results, click the **Add** button and enter the path to your DLL in the field labeled **Executable**. Enter your extension in the **Extension** field (.dwm). Select the radio button labeled **All Verbs** so your DLL will handle all of the page request types. Make sure that **Script Engine** is checked and click the **OK** button. Now, as far as your Web server is concerned, all page requests that end with .dwm will be sent to your DLL.

There are several things you now need to keep in mind while developing your DLL. First, all actions must use the complete file name and path from the Web server root. Thus, if you place a file named Signin.dwm at the Web root, the corresponding action in your DLL will have a PathInfo entry of /Signin.dwm. You can't omit the extension. Next, be aware that even if you'd like to simply handle an action that redirects the caller to another page, you'll still need to create a corresponding .dwm file even if you decide to leave it empty.

Be aware that there is no way to stream image or other binary data out to a Web page when using custom extensions. If you stream charts or other binary data, do so in a separate DLL.

Finally, the MDWeb components have to be slightly modified to work with custom extensions. These modifications are included in the MDWeb components accompanying this article.

— Dr Mark Brittingham

```
    property IntValues[const Name: string]: Integer
      read GetIntValue write SetIntValue;
  end;

procedure Register;
procedure MergeStrings(Dest, Source: TStrings);

implementation

procedure MergeStrings(Dest, Source: TStrings);
var
  I, DI: Integer;
begin
  for I := 0 to Source.Count - 1 do begin
    try
      if Pos('=', Source[I]) > 1 then
        begin
          DI := Dest.IndexOfName(Source.Names[I]);
          if DI > -1 then
            Dest[DI] := Source[I]
          else
            Dest.Add(Source[I]);
        end
      else
        if (Dest.IndexOf(Source[I]) = -1) then
          Dest.Add(Source[I]);
    except
    end;
  end;
end;

procedure TMDSessionMgr.CreateSessionObject(
  inUID: Integer = 1);
begin
  SessNode := SessTree.CreateNode(inUID);
  SessID := SessNode^.ID;
end;

function TMDSessionMgr.FindSessionObject: Boolean;
begin
  SessNode := SessTree.FindNodeByID(SessID);
  Result := (SessNode <> nil);
end;

procedure TMDSessionMgr.DoTagEvent(Tag: TTag;
  const TagString: string; TagParams: TStrings;
  var ReplaceText: string);
var
  astr : string;
  InStream : TStream;
  Month, Day, Year : Word;
begin
  aStr := Uppercase(TagString);
  if (aStr = 'SESSID') then
    ReplaceText := IntToStr(SessID)
  else if (aStr = 'INCLUDE') then
    begin
      try
        InStream := TFileStream.Create(TagParams[0],
                      fmOpenRead + fmShareDenyWrite);
        if InStream <> nil then
          try
            ReplaceText := ContentFromStream(InStream);
          finally
            InStream.Free;
          end;
      except
        ReplaceText := 'Error in file include! ';
      end;
    end
  else if (aStr = 'CURDATE') then
    ReplaceText := DateToStr(Date)
  else if (aStr = 'RAND') then
    begin
      Randomize;
      ReplaceText := IntToStr(Random(1000));
    end
  // If you place the STOREFORMVARS tag in a Form-response
  // page, it will automatically copy all form responses
  // to the visitor's session.
  else if (aStr = 'STOREFORMVARS') then
    try
      if (SessNode <> nil) then
        MergeStrings(SessNode^.Data,
          Dispatcher.Request.ContentFields);
      ReplaceText := ' ';
    except
    end
  else
    // User can implement his own global subsets.
    inherited DoTagEvent(Tag, TagString,
      TagParams, ReplaceText);
end;

function TMDSessionMgr.HandleTag(const TagString: string;
  TagParams: TStrings): string;
var
  astr : string;
  i : Integer;
begin
  astr := Inherited HandleTag(TagString, TagParams);
  if (astr ='') then
    begin
      astr := '<#' + TagString;
      for i := 0 to TagParams.Count - 1 do
        astr := astr + ' ' + TagParams[i];
      astr := astr + '>';
    end;
  Result := astr;
end;

function TMDSessionMgr.ContentFromFile(
  filename: string): string;
var
  InStream: TStream;
begin
  InStream := TFileStream.Create(filename,
                fmOpenRead + fmShareDenyWrite);
  if InStream <> nil then
    try
      Result := ContentFromStream(InStream);
    finally
      InStream.Free;
    end;
  else
    Result := '<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML ' +
      '3.2 Final//EN"><HTML><HEAD><TITLE>Error</TITLE>' +
      '</HEAD>'<BODY>Error: The page you have requested ' +
      'cannot be found.  Please report this error to the' +
      ' webmaster of this site!</BODY></HTML>';
end;

procedure TMDSessionMgr.SetValue(
  const Name, Value: string);
begin
  SessNode^.Data.Values[Name] := Value;
end;

function TMDSessionMgr.GetValue(
  const Name: string): string;
begin
  Result := SessNode^.Data.Values[Name];
end;

procedure TMDSessionMgr.SetIntValue(const Name: string;
  const Value: Integer);
begin
  SessNode^.Data.Values[Name] := IntToStr(Value);
end;
```

```
function TMDSessionMgr.GetIntValue(
  const Name: string): Integer;
begin
  Result := StrToIntDef(SessNode^.Data.Values[Name], -1);
end;

procedure Register;
begin
  RegisterComponents('Internet', [TMDSessionMgr]);
end;

end.
```

End Listing One

Begin Listing Two — *TMDPageProducer* Object

```
unit MDPageProducer;

interface

uses
  Windows, SysUtils, HTTPApp, Classes, MDSessMgr;

type
  TMDPageProducer = class(TPageProducer)
  private
    FSessionMgr : TMDSessionMgr;
  protected
    function HandleTag(const TagString: string;
      TagParams: TStrings): string; override;
  public
    constructor Create(AOwner: TComponent); override;
    function Content: string; override;
  end;

procedure Register;

implementation

constructor TMDPageProducer.Create(AOwner: TComponent);
var
  I: Integer;
  Component: TComponent;
begin
  // This MDPageProducer Create routine will look through
  // all the components already created and, if one is a
  // SessionMgr, capture a pointer to it. This way we don't
  // have to manually chain the Session handling on to the
  // normal page handling. This is why it's critical that
  // the SessionMgr comes before any MDPageProducers in
  // the creation order of the WebModule.
  inherited Create(AOwner);
  if Owner <> nil then
    for I := 0 to Owner.ComponentCount - 1 do begin
      Component := Owner.Components[I];
      if Component is TMDSessionMgr then
        FSessionMgr := TMDSessionMgr(Component);
    end;
end;

function TMDPageProducer.HandleTag(const TagString: string;
  TagParams: TStrings): string;
var
  astr : string;
  i : Integer;
begin
  // We override this function because we don't want the
  // default behavior. By default a PageProducer leaves a
  // blank if it can't handle a tag. However, we want the
  // tags to survive so that the Session Manager has a
  // chance to handle any that we don't. So, I just undo
  // the erase that is normally done here. Also, I have a
  // philosophical problem with not leaving the tag in; it
  // makes it harder to see what work remains to be done,
  // and easier to overlook things during development.
  astr := Inherited HandleTag(TagString, TagParams);
  if (astr = '') then
    begin
      astr := '<#' + TagString;
      for i := 0 to TagParams.Count - 1 do
        astr := astr + ' ' + TagParams[i];
      astr := astr + '>';
    end;
  Result := astr;
end;

function TMDPageProducer.Content: string;
begin
  // Key part - after passing the content through the
  // normal WebBroker channels(the inherited Content),
  // I pass it through the session object as well.
  Result := Inherited Content;
  if (FSessionMgr <> nil) then
    Result := FSessionMgr.ContentFromString(Result);
end;

procedure Register;
begin
  RegisterComponents('Internet', [TMDPageProducer]);
end;

end.
```

End Listing Two

By Jason Perry

Exploiting SQL Server 7 DMO

Part II: A Database Tool and a Security Object

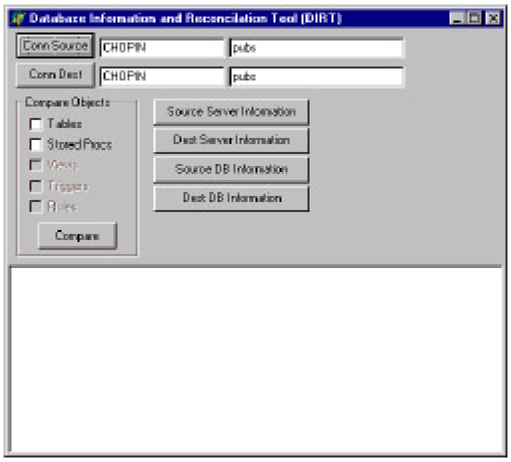
In Part I of this two-part series, we looked at the basics of SQL Server DMO objects. We also looked at a script-writing tool for SQL developers, named SSB (SQL Script Builder).

In this installment, we'll look at DIRT, or Database Information and Reconciliation Tool, for cross-database comparisons. We'll also look at a simple security object, and demonstrate how it can be used in application development.

Building a Database Information and Reconciliation Tool (DIRT)

How do you keep your test database in sync with your production database? How do you keep your replicated servers in sync? What if you have multiple environments for the same database? SQL-DMO can be used to compare server objects and look for inconsistencies. In this wimpy demo, I'll show you how to use the *Tables*, *Columns*, and *StoredProcedures* collections and their corresponding *Table* and *StoredProc* objects to build an analysis utility to compare server objects for consistency. I will also demonstrate how the *SQLServer* and *Database* objects can be used to report on database sizes and server activity.

The first thing I did was write a method to connect to the named source server and named destination server:



CHOPIN pubs
CHOPIN pubs

Figure 1: DIRT disconnected.

```
function ConnectSource(sServerName:
                          string): Boolean;
function ConnectDest(sServerName:
                        string): Boolean;
```

The argument for each is the server name for the source and for the destination. The methods create two separate server objects to be referenced as public members of the class. Next, I have two methods that create and store a reference to two database objects:

```
function ConnectDBSource(sDBName:
                            string): Boolean;
function ConnectDBDest(sDBName:
                          string): Boolean;
```

The argument is the name of the database for each one. This is pretty much a no-brainer. It's almost exactly what I did to create the database objects in SSB.

DIRT uses the same ODBC connection you created in SSB. Start it up and you're presented with a screen that has a Conn Source and a Conn Dest button (see Figure 1). Type in the names of the server and the database. Click the corresponding Conn... button to create the server and database objects.

If the objects connected correctly, you will hear a light "beep."

SQLServer and *Database* Attributes

The *SQLServer* and *Database* objects have many attributes that can help you observe the status of your server and databases. I exploited some of these attributes in DIRT. To see the *SQLServer* object attributes, click each of the ...Server Information buttons (see Figure 2). These call a small method that adds lines to the *TMemo*. They use the various attributes of the *SQLServer* object (see Figure 3).

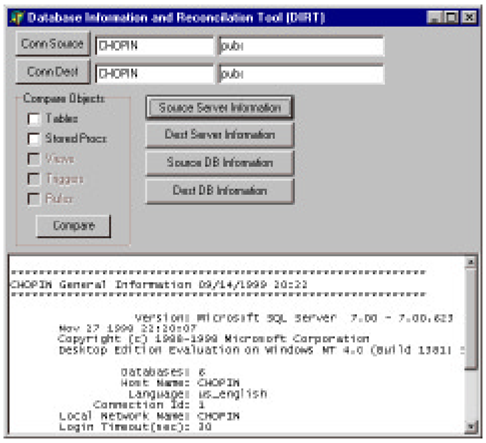
See the SQL Server books online for an accurate definition of each attribute. Where practical, you could write a simple comparison routine to make sure each server has similar attributes.

Next, click each of the ...DB Information buttons (see Figure 4). These, like the server info method, simply add lines to the *TMemo* about the database sizes (see Figure 5).

Running out of disk space in a database can be catastrophic. You could write a service to poll the database size and e-mail you if it reaches a certain threshold. The possibilities are endless.

The *Tables*, *Columns*, and *StoredProcedures* Collections

Just like SSB, DIRT uses the *Tables*, *Columns*, and *StoredProcedures* collections to enumerate the *Table*, *Column*, and *StoredProcedure* objects for a specific database. I wrote a couple of simple comparison routines to take the source and destination databases and compare some entity attributes (see Figure 6). This is an invaluable technique for reconciling the differences between two databases. Nothing is worse than having a test database that works, and a production one that doesn't.



```
*************************************************************
CHOPIN General Information 09/14/1999 20:22
*************************************************************

                Version: Microsoft SQL Server  7.00 - 7.00.623
    Nov 27 1998 22:20:07
    Copyright (c) 1988-1998 Microsoft Corporation
    Desktop Edition Evaluation on Windows NT 4.0 (Build 1381: S

              Databases: 6
              Host Name: CHOPIN
               Language: us_english
          Connection Id: 1
    Local Network Name: CHOPIN
    Login Timeout(sec): 30
```

Figure 2: DIRT server information.

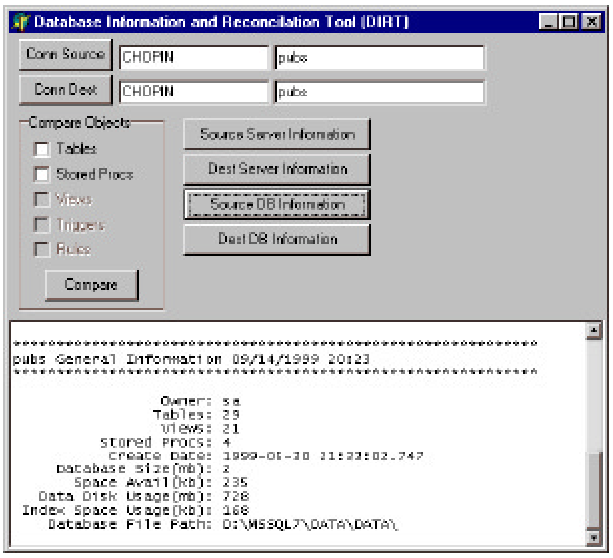
```
procedure TboDirt.ServerInfo(oServer: _SQLServer);
begin
  if Assigned(CompareUI) then begin
    Banner(oServer.Name + ' General Information');
    CompareUI.Add('                  Version: ' +
      oServer.VersionString);
    CompareUI.Add('                Databases: ' +
      IntToStr(oServer.Databases.Count));
    CompareUI.Add('                Host Name: ' +
      oServer.HostName);
    CompareUI.Add('                 Language: ' +
      oServer.Language);
    CompareUI.Add('            Connection Id: ' +
      IntToStr(oServer.ConnectionId));
    CompareUI.Add('       Local Network Name: ' +
      oServer.NetName);
    CompareUI.Add('       Login Timeout(sec): ' +
      IntToStr(oServer.LoginTimeout));
    CompareUI.Add('       Query Timeout(sec): ' +
      IntToStr(oServer.QueryTimeout));
    CompareUI.Add('     Blocking Timeout(ms): ' +
      IntToStr(ord(oServer.BlockingTimeout)));
    CompareUI.Add('       Command Terminator: ' +
      oServer.CommandTerminator);
    CompareUI.Add('    This Application Name: ' +
      oServer.ApplicationName);
    CompareUI.Add('      Auto Reconnect Flag: ' +
      IntToStr(ord(oServer.AutoReconnect)));
    CompareUI.Add('        Default Null Flag: ' +
      IntToStr(ord(oServer.AnsiNulls)));
    CompareUI.Add('      Network Packet Size: ' +
      IntToStr(oServer.NetPacketSize));
  end;
end;
```

Figure 3: Adding lines to the Memo component.

The source is simple, as shown in Listing One (beginning on page 18). First, we spin through the source *Tables* collection and get a reference to each table object. Next, we spin through the *Tables* collection of the destination server object. I want to make an important note here. Notice the statement:

```
DB_Dest.Tables.Item(lcv2, DB_Dest);
```

I could have passed the physical name of the source table (*oSourceTable.Name*) here to return the *Table* object. The problem is that if the table isn't in the *Tables* collection on the destination database, it will raise an exception that it is not found. Because I wanted a more graceful way of reporting table discrepancies, I spun through the *Tables* collection looking for the same named table. Now I can report on the discrepancy by adding a line to the *TMemo*.



```
*************************************************************
pubs General Information 09/14/1999 20:23
*************************************************************

                  Owner: sa
                 Tables: 29
                  Views: 21
           Stored Procs: 4
            Create Date: 1999-01-30 21:22:02.747
      Database Size(mb): 2
        Space Avail(kb): 235
   Data Disk Usage(mb): 728
 Index Space Usage(kb): 168
    Database File Path: D:\MSSQL7\DATA\DATA\
```

Figure 4: DIRT database information.

```
procedure TboDirt.DBInfo(oDB: _Database);
begin
  if Assigned(CompareUI) then begin
    Banner(oDB.Name + ' General Information');
    CompareUI.Add('                Owner: ' + oDB.Owner);
    CompareUI.Add('               Tables: ' +
      IntToStr(oDB.tables.Count));
    CompareUI.Add('                Views: ' +
      IntToStr(oDB.Views.Count));
    CompareUI.Add('         Stored Procs: ' +
      IntToStr(oDB.StoredProcedures.Count));
    CompareUI.Add('          Create Date: ' +
      oDB.CreateDate);
    CompareUI.Add('    Database Size(mb): ' +
      IntToStr(oDB.Size));
    CompareUI.Add('      Space Avail(kb): ' +
      IntToStr(oDB.SpaceAvailable));
    CompareUI.Add('   Data Disk Usage(mb): ' +
      FloatToStr(oDB.DataSpaceUsage));
    CompareUI.Add(' Index Space Usage(kb): ' +
      FloatToStr(oDB.IndexSpaceUsage));
    CompareUI.Add('   Database File Path: ' +
      oDB.PrimaryFilePath);
  end;
end;
```

Figure 5: Adding information about database sizes.

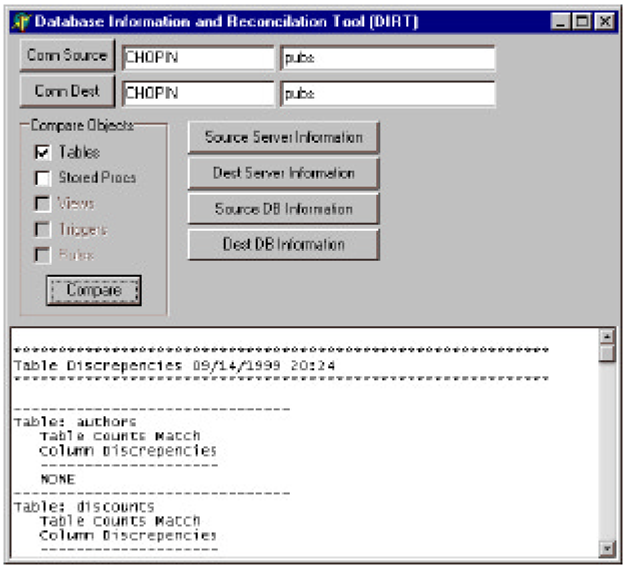
Lastly, I spin through the *Columns* collection for each table (don't forget the SQL-DMO object hierarchy) and compare the source and destination table column attributes. This is a common place for DBAs to make errors.

You can do other things here, as well. Compare the table scripts between tables to make sure they were identically created, that indexes are the same, and to check constraints, rules, etc. The power is awesome!

Building a COM/SQL-DMO Role-based Security Object

You've seen a couple of neat tools that can be written using SQL-DMO objects. What about using them in application development? In this small demo, I'll create a simple COM-based security object. It interrogates the *SQLServer* object to see if a given login is in a particular role, using the *DatabaseRoles* collection. Some additional methods are added to return the users' roles, available roles, and the ability to add or remove a person to/from a particular role.

First, I created a type library and Automation object named *TSQLDMO_Security*. Avoiding a lengthy lesson in the Delphi COM Expert, the Automation object has methods named *GetRoles*, *GetUsers*, *IsUserInRole*, *GetUserRoles*, *AddUserToRole*, *RemoveUserFromRole*, and *Login* (see Figure 7).



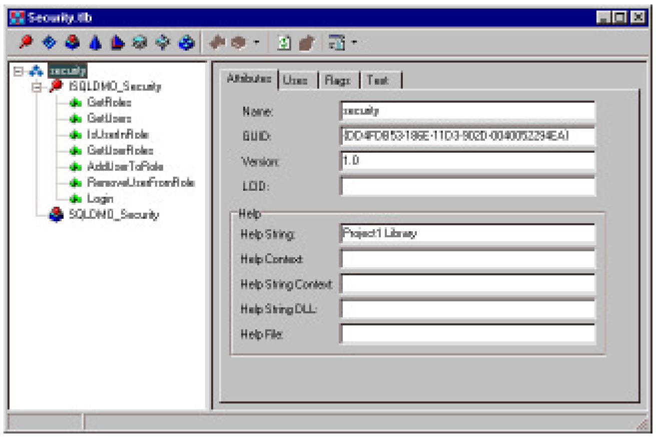
Conn Source	CHOPIN	pubs
Conn Dest	CHOPIN	pubs

Compare Objects: [✓ Tables] Stored Procs Views Triggers Rules

```
*************************************************************
Table Discrepencies 09/14/1999 20:24
*************************************************************

--------------------------------
Table: authors
   Table counts Match
   Column Discrepencies
   --------------------
   NONE
--------------------------------
Table: discounts
   Table counts Match
   Column Discrepencies
   --------------------
```

Figure 6: Comparing tables with DIRT.



```
security
  ISQLDMO_Security
    GetRoles
    GetUsers
    IsUserInRole
    GetUserRoles
    AddUserToRole
    RemoveUserFromRole
    Login
  SQLDMO_Security
```

Name:	security
GUID:	{DD4FD853-186E-11D3-902D-0040052234EA}
Version:	1.0
LCID:	
Help String:	Project1 Library
Help Context:	
Help String Context:	
Help String DLL:	
Help File:	

Figure 7: The SQLDMO_Security class in Delphi's Type Library editor.

In my sample unit (oSecurity.pas), I overrode the *Initialize* method (see Figure 8).

I did this so the *SQLServer* object would be created the first time it was called, and each time additional users called it. Now each user has its own server connection. If all your users are connecting through a single application, you may want to make the *SQLServer* object a singleton to save on resources and increase performance. Note: Set that *ApplicationName* and your DBAs will love you.

All you have to do on the application side is create the Automation object and call the *Login* method:

```
// A login routine. Wimpy.
function TSQLDMO_Security.Login(const sServer,
  sDatabase, sLogin, sPWD: WideString): WordBool;
begin
  oServer.Connect(sServer, sLogin, sPWD);
  Result := oServer.VerifyConnection(
    SQLDMOConn_ReconnectIfDead);
end;
```

Simply pass in the name of the server, the database, the login, and the password of the connection. If the login was successful, it will return True.

Next, let's talk about the *GetRoles* and *GetUsers* implementation, shown in Figure 9.

```
procedure TSQLDMO_Security.Initialize;
begin
  inherited Initialize;
  if Assigned(oServer) then
    oServer := nil;
  else
    begin
      oServer := getServer;
      oServer.Set_QueryTimeout(5);
      oServer.Set_LoginSecure(True);
      oServer.Set_ApplicationName(
        'SQLDMO Security Object');
    end;
end;
```

Figure 8: Overriding the *Initialize* method.

```
// Return a database object by name.
function TSQLDMO_Security.getDB(
  sDBName: string): _Database;
begin
  Result := oServer.Databases.Item(sDBName, '');
  if not Assigned(Result) then
    raise Exception.Create('The database ' +
      sDBName + ' was not found.');
end;

// Get the roles collection for a specified database.
function TSQLDMO_Security.GetRoles(
  const sDBName: WideString): OleVariant;
begin
  Result := getDB(sDBName).DatabaseRoles;
end;

// Get a users collection for a specified database.
function TSQLDMO_Security.GetUsers(
  const sDBName: WideString): OleVariant;
begin
  Result := getDB(sDBName).Users;
end;
```

Figure 9: Implementing *GetRoles* and *GetUsers*.

I created a private *getDB* method to return the database object by name. Very simply, once the Automation object is created, you call its *GetRoles* or *GetUsers* collection with the argument of the database name. They each return an OleVariant that represents the *DatabaseRoles* and *Users* collections. Figure 10 shows some of the code from the test application that fills a *TListBox* with the resulting collections.

```
procedure TForm1.FillUsers;
var
  o : OleVariant;
  lcv : Integer;
begin
  o := Security.GetUsers(txtDatabase.Text);
  lstUsers.Items.Clear;
  for lcv := 1 to o.Count do
    lstUsers.Items.Add(o.Item(lcv).Name);
end;

procedure TForm1.FillUserRoles(sLogin: WideString);
var
  o : IStrings;
  lcv : Integer;
begin
  o := Security.GetUserRoles(sLogin, txtDatabase.Text);
  if Assigned(o) then
    begin
      lstUserRoles.Items.Clear;
      for lcv := 0 to o.Count-1 do
        lstUserRoles.Items.Add(o.Item[lcv]);
    end
  else
    raise Exception.Create('User not found in database');
end;
```

Figure 10: This fills a *TListBox* with the resulting collections.

```
function TSQLDMO_Security.GetUserRoles(const Login,
  sDBName: WideString): IStrings;
var
  lcv : Integer;
  oUser : _User;
  oStrings : TStrings;
  oRoles : NameList;
  lFound : Boolean;
  oDB : _Database;
  oSA : IStrings;
begin
  lFound := False;
  for lcv := 1 to getDB(sDBName).Users.Count do begin
    oDB := getDB(sDBName);
    if Pos(Login, oDB.Users.Item(Login).Login) > 0 then
      begin
        oUser := oDB.Users.Item(Login);
        oRoles := oUser.ListMembers;
        lFound := True;
        Break;
      end
    else
      begin
        lFound := False;
        Continue;
      end;
  end;
  if lFound then
    begin
      oStrings := TStringList.Create;
      GetOleStrings(oStrings, OSA);
      for lcv := 1 to oRoles.Count do
        oSA.Add(oRoles.Item(lcv));
      Result := oSA;
    end
  else
    Result := nil;
end;
```

Figure 11: Implementing the *GetUserRoles* method using the *IStrings* interface.

Looks familiar doesn't it? Just spin through the collections and exploit the attributes you need, and *voilà*! You now have a list of *DatabaseRoles* and *Users*.

To help convince you that the SQL-DMO objects are flexible, I implemented the *GetUserRoles* method using the *IStrings* interface, instead of the "built-in" collection mechanism (see Figure 11).

This method will return an *IStrings* of roles for a specific login and database. What I did was spin through the *Users* collection to get the *User* object for the specified *Login*. If I find a user, I return a *NameList* collection of the roles the user is a member of. As an interesting twist, I implement the *IStrings* interface using *GetOleStrings*, and fill it up by spinning through the *NameList* collection. Now the test application can treat the list as a *TStrings* collection.

Now for the last methods — *IsUserInRole*, *AddUserToRole*, and *RemoveUserFromRole* (see Figure 12). These methods are simple. They each take the arguments *Login*, *Role*, and *Database Name*.

What's important here are the *IsMember* method of the *User* object, and the *AddMember* and *DropMember* methods of the *DatabaseRoles* object. The *IsUserInRole* method gets the specified database and the specified user in the database, and asks the question: "Is this user in this role?" The *Add*/*Drop* methods get the specified database and the specified *Role* object, and invoke the *AddMember*/*DropMember* methods with the **login** argument. Could this be any easier?

So, what good is an Automation object without a test application? Well, I included one of those too (see Figure 13).

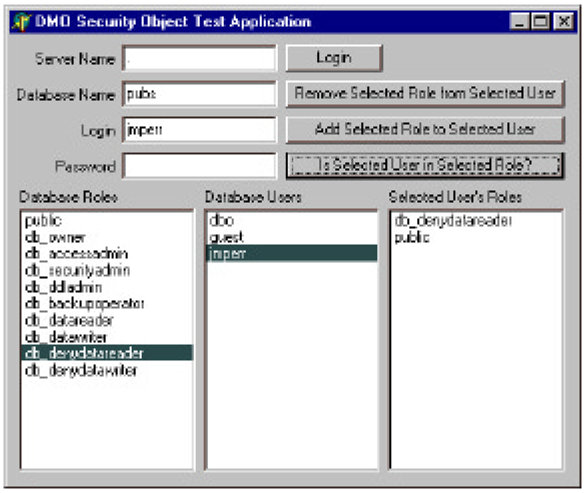
Simply fill out the pertinent server information and click **Login**. Select a database role and a user. In the far right-hand side, *TListBox* will show a list of roles that the person has. Click the **Add Selected Role to Selected User** and **Remove Selected Role from Selected User** buttons, and watch the user role list. Open a copy of Enterprise Manager to confirm that the role was added. The **Is Selected User in Selected Role?** button tests whether ... never mind — it's obvious.

```
// So, is this user login a member of the specified role?
function TSQLDMO_Security.IsUserInRole(
  const Login, Role, sDBName: WideString): WordBool;
begin
  Result :=
    getDB(sDBName).Users.Item(Login).IsMember(Role);
end;

// Add a user to a role.
procedure TSQLDMO_Security.AddUserToRole(
  const Login, Role, sDBName: WideString);
begin
  getDB(sDBName).DatabaseRoles.Item(Role).AddMember(Login);
end;

// Remove a user from a role.
procedure TSQLDMO_Security.RemoveUserFromRole(
  const Login, Role, sDBName: WideString);
begin
  getDB(sDBName).DatabaseRoles.Item(Role).
    DropMember(Login);
end;
```

Figure 12: The *IsUserInRole*, *AddUserToRole*, and *RemoveUserFromRole* methods.


DMO Security Object Test Application
Login
Remove Selected Role from Selected User
Add Selected Role to Selected User
Is Selected User in Selected Role?

Server Name: .
Database Name: pubs
Login: jmperr
Password:

Database Roles: public, db_owner, db_accessadmin, db_securityadmin, db_ddladmin, db_backupoperator, db_datareader, db_datawriter, db_denydatareader, db_denydatawriter

Database Users: dbo, guest, jmperr

Selected User's Roles: db_denydatareader, public

Figure 13: A test application.

Conclusion

SQL-DMO is vast. This series has touched on the major collections and objects to help get you started on your enterprise tool. You can rebuild indexes, check for page integrity, add indexes, change object attributes, replicate your servers, and more. Writing smart tools to manage routine database maintenance, such as status reporting and user security issues, are now a simple chore with the SQL-DMO objects. Hopefully, I've helped open a new world of possibilities for managing your enterprise SQL Servers. ∆

Resources

- Compass Technology Management: http://www.compass.net
- Object-oriented JAVA Enterprise Architecture Experts: OOP.COM, http://www.oop.com
- Microsoft SQL-DMO FAQ: http://msdn.microsoft.com/library/techart/msdn_dmoovrvw.htm
- Object-oriented Programming in Delphi and Java: http://www.oop.com

Jason 'Wedge' Perry is a system architect for OOP.COM in Chesapeake, VA. Prior to accepting this position, Wedge was a self-employed consultant in development positions ranging from grunt programmer to system architect. In his spare time, Wedge races a Kawasaki KX250 moto-cross motorcycle for the Elizabeth City MX Club.

Begin Listing One — DIRT

```
procedure TboDirt.CompareTables;
var
  lcv, lcv2, lcv3 : Integer;
  oSourceTable, oDestTable : _Table;
  lFound, lColErr : Boolean;
begin
  if not(Assigned(SQLDMO_Source)) or
     not(Assigned(SQLDMO_Dest)) then
    Exit;

  Banner('Table Discrepencies');
  for lcv := 1 to DB_Source.Tables.Count do begin
    // Get the  first table's name.
    oSourceTable := DB_Source.Tables.Item(lcv, DB_Source);
    oDestTable := nil;
    // Update the UI;
    if Assigned(CompareUI) then
      CompareUI.Add('------------------------------');
    if Assigned(CompareUI) then
      CompareUI.Add('Table: ' + oSourceTable.Name);
    // if Assigned(CompareUI) then
    //   Application.Processmessages;
    // See if the number of tables is consistent.
    if DB_Source.Tables.Count > DB_Dest.Tables.Count then
      begin
        if Assigned(CompareUI) then
          CompareUI.Add(
            '    ** There are additional tables in Source.');
      end
    else if DB_Source.Tables.Count <
            DB_Dest.Tables.Count then
      begin
        if Assigned(CompareUI) then
          CompareUI.Add(
            '    ** There are missing tables in Source.');
      end
    else if DB_Source.Tables.Count =
            DB_Dest.Tables.Count then
      begin
        if Assigned(CompareUI) then
          CompareUI.Add('   Table Counts Match');
      end;
    // Look for that table name in destination database.
    // Note: The table could be out of order. If it isn't
    // found, the table should be noted as missing.
    lFound := False;
    for lcv2 := 1 to DB_Dest.Tables.Count do begin
      oDestTable := DB_Dest.Tables.Item(lcv2, DB_Dest);
      // In case the table doesn't exist.
      if not (Assigned(oDestTable)) then
        begin
          if Assigned(CompareUI) then
            CompareUI.Add('    ** Missing Table: ' +
                          oSourceTable.Name);
          Break;
        end;
      // Looking for the same table.
      if UpperCase(oSourceTable.Name) =
         UpperCase(oDestTable.Name) then
        begin
          lFound := True;
          // Check each column.
          if Assigned(CompareUI) then
            CompareUI.Add('    Column Discrepencies');
          if Assigned(CompareUI) then
            CompareUI.Add('    -------------------');
          lColErr := False;
          for lcv3 := 1 to
              oSourceTable.Columns.Count do begin
            // Test the data type.
            if oSourceTable.Columns.Item(lcv3).Datatype <>
               oDestTable.Columns.Item(lcv3).Datatype then
              begin
                if Assigned(CompareUI) then
                  CompareUI.Add('    Datatype: ' +
                    oSourceTable.Columns.Item(lcv3).Name +
                    oSourceTable.Columns.Item(lcv3).
                      DataType + ' ' +
                    oDestTable.Columns.Item(lcv3).DataType);
                lColErr := True;
              end;
            // Test the physical data type.
            if oSourceTable.Columns.Item(lcv3).
                 PhysicalDatatype <>
               oDestTable.Columns.Item(lcv3).
                 PhysicalDatatype then
              begin
                if Assigned(CompareUI) then
                 CompareUI.Add('    PhysicalDatatype: ' +
                  oSourceTable.Columns.Item(lcv3).Name +
                  oSourceTable.Columns.Item(lcv3).
                    PhysicalDatatype + ' ' +
                  oDestTable.Columns.Item(lcv3).
                    PhysicalDatatype);
                lColErr := True;
              end;
            // Test the AllowNulls property.
            if oSourceTable.Columns.Item(lcv3).AllowNulls<>
              oDestTable.Columns.Item(lcv3).AllowNulls then
              begin
                if Assigned(CompareUI) then
                  CompareUI.Add('   AllowNulls: ' +
                    oSourceTable.Columns.Item(lcv3).Name +
                    IntToStr(ord(Boolean(oSourceTable.
                    Columns.Item(lcv3).AllowNulls))) + ' '+
                    IntToStr(ord(Boolean(oDestTable.
                    Columns.Item(lcv3).AllowNulls))));
                lColErr := True;
              end;
            // Test the Length property.
            if oSourceTable.Columns.Item(lcv3).Length <>
               oDestTable.Columns.Item(lcv3).Length then
              begin
                if Assigned(CompareUI) then
                  CompareUI.Add('   Length: ' +
                    oSourceTable.Columns.Item(lcv3).Name +
                    IntToStr(oSourceTable.Columns.Item(
                    lcv3).Length) + ' ' + IntToStr(
                    oDestTable.Columns.Item(lcv3).Length));
                  lColErr := True;
              end;
            Break;
          end;
          // If no errors, then put a NONE in.
          if not lColErr then
            if Assigned(CompareUI) then
              CompareUI.Add('   NONE');
        end;
    end;
    if not lFound then
      // Update the UI;
      if Assigned(CompareUI) then
        CompareUI.Add('   **Table: ' + oSourceTable.Name +
          ' Not found in destination database.');
  end;
end;
```

End Listing One

By Ron Loewy

Palm Conduits

Part II: Writing a Sample ToDo Application

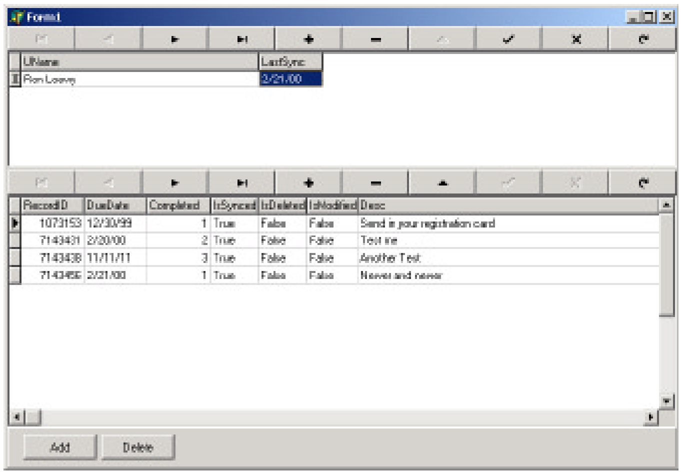
In Part I of this series, we introduced the concept of programming for the Palm handheld device and the use of conduits, which performs the data synchronization between the Palm and the PC. We also introduced EHAND Connect, a COM-based product that allows you to write conduits with every Windows development tool that can create automation objects.

In this half of the series, I'll demonstrate the use of EHAND Connect to create a conduit in Delphi by writing a simple ToDo application that will store information from the Palm device ToDo application to a Paradox database. We will also write a conduit that synchronizes between the Paradox database and the Palm device.

About the PC ToDo Application

The goal of this article is to discuss conduit creation. Therefore, the ToDo application we're going to write will be a primitive application that will simply allow us to create data for synchronization. A lot of the operations that a well-designed ToDo application will perform behind the scenes will be exposed in this application.

For example, the database we will use will have an IsDeleted field for every row. In a real application, when the user chooses to delete a row, this field will be set to True, and the data won't be displayed to the user. The record will remain in the database until it is synchronized with the Palm device; otherwise, we won't know that the record (assuming it exists on the Palm) needs to be removed. In our application, however, all the records are always displayed, including the "deleted" ones that are only marked as deleted for synchronization purposes.



UName	LastSync
Ron Loewy	2/21/00

RecordID	DueDate	Completed	IsSynced	IsDeleted	IsModified	Desc
1073153	12/30/99	1	True	False	False	Send in your registration card
7143431	2/20/00	2	True	False	False	Test me
7143438	11/11/11	3	True	False	False	Another Test
7143456	2/21/00	1	True	False	False	Never and never

Figure 1: Our sample PC database application.

The ToDo Database

I used the Database Desktop to create a simple, two-table database (see Figure 1). I later used the Control Panel BDE Administrator applet to create a BDE alias called "PALMSAMP" to point to this database.

The sample tables are included with the source archive of this article (see end of article for download details). You can unzip it to a directory of your choice, and set the alias to point to that directory. The database includes a Users table, which has a LastSync field. The database can be used by many users and we don't want to synchronize the ToDo list of one user with the information of another user. (In reality, the synchronization code I wrote does update this table, but it always assumes that all the records in the Entries table belong to the synchronized device, and will always synchronize all of them. In a real-life application, you'll define the Entries table as a detail table with the Users table as its master table.)

The Entries table is the interesting data storage for our application. It includes the fields that describe the ToDo information (DueDate, Completed, Description, and Notes) and three flags: IsSynced determines if the record was ever synchronized with the Palm device, IsDeleted determines if the user deleted the record on the PC and it needs to be deleted on the Palm device, and IsModified determines if the record has been modified on the PC. The RecordID field is the unique identifier of the record in the database. The value in this field is used to link a record on the PC with a record on the Palm device.

Synchronization Strategy

When we need to synchronize databases on different devices, we need to determine a strategy for clash resolution. Consider the case where the same record is modified on the PC: It's also modified on the Palm device before synchronization can be performed. What should our strategy be when we come to synchronize the databases? Should the PC record prevail, or should the Palm record prevail?

The situation could be even more complicated. What happens to a record that was modified on the Palm device, but was deleted on the PC? Should we remove the record from both devices, or should we update the record on the PC and "undelete" it?

The truth is that there are no absolute answers. What you decide to do is the way your application resolves these conflicts. Taking the first scenario (the record modified on both platforms before synchronization), we can solve the problem by adding a ModificationTime field that is updated whenever the record is modified. In this case, the more recently modified record will prevail. Another solution (the one I am taking in the sample application) is to opt for handheld modification over PC modification; if both records that need to be reconciled have been modified, I prefer the Palm modification.

In the sample application, I decided to ignore the archive bit that can be assigned to a Palm record. When this bit is turned on, the PC needs to store a copy of the record for archival purposes during synchronization. However, the size of the code that is needed to illustrate basic synchronization that takes care of essential stuff, like record addition, modification, and purging, is large enough as it is, and it seemed reasonable to try and keep the size of the code small since this is an article, not a book about Palm programming. The strategy I decided to follow is summarized in the table in Figure 2.

Supporting Cast

The file SyncData.pas includes the definition of the *TToDoRecord* class. This class represents data that is stored in memory during the synchronization process. We will keep two memory databases: one for the "interesting" PC records, and one for all the records in the Palm database. Every record in these "memory" databases will be represented by a *TToDoRecord* object.

A *TToDoRecord* object has properties for the information related to the ToDo database — *DueDate*, *Completed*, *Description*, and *Note*. It also has a field called *Operation* that includes the synchronization operation our conduit decides to perform on the record. The available options are *opAddPalm*, *opModifyPalm*, *opDeletePalm*, *opAddPC*, *opModifyPC*, *opDeletePC*, and *opNoOp*. The last one represents a "No Operation" option.

The class also has a *RecordID* property. This is the unique key that allows us to match records between the PC and Palm databases.

The *PalmRecord* property points to an EHAND Connect Palm Data Record interface and provides access to the methods that are available for this record on the Palm. The *PCRecNo* property is used to provide a cursor pointer on the PC database for a record created from entry on the PC. This allows us to position the cursor in the Paradox table to the specific record, if we need to modify some of the fields in the record.

The class provides some utility methods, including *SetPalmRecord*, which sets the attributes and properties from an EHAND Connect Palm Data Record interface pointer, and *SetPCRecord*, which sets these attributes and properties from the current record of a *TTable* instance.

Synchronization in Action

The project ToDoSync.dpr was created using Delphi's ActiveX Library wizard. It includes the automation object class, named *DIConduit*, that hosts our code.

The file, SyncUnit.pas, contains the synchronization code and implements the strategy we discussed earlier. I created an Automation object (using Delphi's Automation Object wizard) and used the Type Library editor to add the *OpenConduit* method. This is the method called by EHAND Connect when the conduit code starts the synchronization process.

In *OpenConduit*, we set the variable *ConduitObj* to point to the conduit interface passed by EHAND Connect. We call the function *HandleUser*, which checks against, and updates, the Users table. In a real application, the user information would be used to synchronize information only against the user information, but for simplicity's sake, we assume that all the records in the Entries table will be synchronized. I won't discuss *HandleIUser* in this article, but you can inspect the source to see how the information is retrieved from the Palm device.

The next call is to the *DefineSchema* method. This method opens the ToDoDB database on the Palm device, and defines the schema (data structure) of the database. The schema on the Palm device consists of four fields: DueDate, CompletedFlag, Desc, and Note. Notice the use of field type constants (*eDate*, *eByte*, and *eString*) in the call to the *DefineField* method of the conduit interface:

```
if (ConduitObj.OpenDatabase('ToDoDB') <> 0) then begin
  ConduitObj.DefineField('DueDate', eDate, '');
  ConduitObj.DefineField('CompletedFlag', eByte, '');
  ConduitObj.DefineField('Desc', eString, '');
  ConduitObj.DefineField('Note', eString, '');
end;
```

After the schema has been defined, we determine the type of synchronization requested. The options are Do Nothing, Copy from the

Palm record	PC record	Operation
Modified	Nothing	Modify PC Record.
Modified	Modified	Modify PC Record.
Modified	Deleted	Modify PC Record and Undelete it.
Modified	Does not exist	Create a new PC Record.
Does not exist	Modified/Not Synced	Create a new Palm Record.
Nothing	Modified	Modify Palm Record.
Deleted	Modified	Modify Palm Record and Undelete it.
Nothing	Deleted	Delete Record on both platforms.
Deleted	Nothing	Delete Record on both platforms.
Deleted	Deleted	Delete Record on both platforms.
Nothing	Nothing	Nothing to do.

Figure 2: My synchronization strategy.

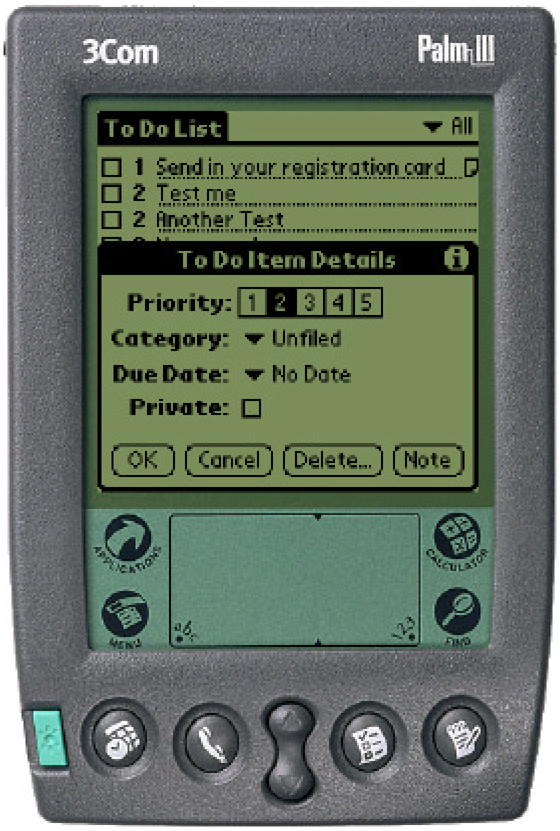


Figure 3: The data in the Palm ToDo List application after synchronization.

PC to the Hand Held device, Copy from the Hand Held device to the PC, or synchronize both. In the sample application, I only implemented the synchronize both option, because it's the most common and most complicated option (the others are sub-sets).

Complete synchronization is performed in the method called *SyncBoth*, which we'll examine in a moment. After synchronization is finished, the database is closed and an entry of our success is written to the synchronization log. The synchronization log can be inspected on the Palm device after the HotSync program finishes the synchronization process.

SyncBoth is the method that performs the actual data synchronization. Let's take a "grand" view of its operation:

One. The "suspect" records are read from the database into a memory structure (held in a *TStringList* named *DatabaseRecords*) using the *DBToMemory* method. Initial suspect operations are assigned to the memory records, e.g. *opModifyPalm*, *opAddPalm*, etc.

Two. All records from the Palm device are read into a memory structure, held in a *TStringList* named *PalmRecords*. The record's attributes are inspected, and an initial operation is assigned to the memory record. At this time, clashes are inspected against the *DatabaseRecords* structure, and we determine the operation that will be taken based on the strategy we discussed previously. If needed, the *DatabaseRecords* record object's operation property is modified. At the end of this stage, we have all the information we need to physically perform the synchronization.

Three. The *UpdatePalmRecords* method is called. It traverses the *PalmRecords* memory structure and, based on the operation property of every record, deletes the record or updates it on the Palm device using the *SetField* and *WriteRec* methods exposed by the EHAND Connect interfaces.

Four. The *UpdateDBRecords* method is called. It traverses the *DatabaseRecords* memory structure and performs the physical operations of updating both the PC and the Palm device as needed. Because we use the *RecNo* property of a *TTable* to position the cursor, we actually perform three passes on the *DatabaseRecords* structure. In the first, we perform the *opAddPalm*, *opDeletePalm*, *opModifyPalm*, and *opModifyPC* operations. In the second pass, we perform the *opDeletePC* operations. In the third, we add new records to the PC.

Notice that during step Four we call the conduit interface's *PurgeDeletedRecs* method. This method physically deletes records in the Palm device that have been marked as deleted earlier. Figure 3 shows the Palm ToDo List application after synchronization.

Final Steps

To activate our conduit code, we need to register it with the HotSync.exe application. After building the project in Delphi and registering it (select **Run | Register ActiveX Server** from Delphi, or run regsvr32.exe from the command line), I used CondCfg.exe, a utility that can be found in the Bin sub-directory of the EHAND Connect installation (C:\program files\ehand\ehand connect on my machine).

Using this utility, you need to associate the conduit with an application installed on the Palm device. We could, of course, associate the conduit with the ToDo application, but this would remove the standard ToDo conduit that ships with the Palm Desktop software. Because we don't really want to replace this conduit, and just want to disable it while we test our code, I chose to associate our conduit with the Calc application (the Palm application that displays a calculator), which does not store data in a database and therefore does not need to synchronize information with a desktop. It is thus perfect for our sample. Obviously, in the real world, you will probably synchronize with a custom application you created and installed on the Palm device, and you will associate the conduit with this application.

At this point (before adding the reference to our conduit code), if you use the ToDo application on the Palm device, I would suggest using HotSync to synchronize your data with the PC. The data we might scramble playing with the conduit code can then be restored later from the PC.

Right-click on the HotSync application's icon in the tray and choose **Custom** from the popup menu. Select the **To Do List** conduit from the list box and click the **Change** button. In the Change HotSync Action dialog box, set the action to **Do Nothing** and click the **OK** button. This will ensure that the records we add, modify, or delete to the Palm database will not find their way into the PC application and we'll be able to restore the real data later.

Activate the CondCfg.exe application mentioned previously and use the **Add** button. Set the **Creator ID** to "calc" (no quotes). This will associate the conduit with the Calc application. In the Conduit Setting dialog box, point the **DLL Name** entry to the file EHConnect.dll in the EHAND Connect installation directory, C:\program files\ehand\ehand connect\EHConnect.dll by default. In the **Class Name** entry, enter the name of the conduit automation object we created — *ToDoSync.DIConduit* in our example.

You're now ready to synchronize using the HotSync application. You can build the sample ToDoDB application that will allow you to edit records in the Paradox table on the PC. Try to add and modify entries on this application, or do the same on the Palm device and synchronize them to see the data transferred between the platforms.

Conclusion

While the official Palm Conduit Development Kit supports only Visual C++, with a little help from EHAND Connect, every COM-enabled development tool can be used to create conduits to exchange and synchronize data between a Palm device and a Windows desktop.

Colin Chapman believed in small, light-weight race cars, but even he would have loved a truck (or, being English, a lorry) to ferry his precious works of art to the Grand Prix circuits, where they dominated the opposition. Think of the Delphi conduit as the on-ramp to the truck — the piece that makes the small Palm device a really useful tool instead of a plaything. Δ

The project files referenced in this article are available on the Delphi Informant Magazine Complete Works CD located in INFORM\00\JUL\DI200007RL.

Ron Loewy is a software developer for HyperAct, Inc. He is the lead developer of eAuthor Help, HyperAct's HTML Help-authoring tool. For more information about HyperAct and eAuthor Help, contact HyperAct at (515) 987-2910, or visit http://www.hyperact.com.

By Cary Jensen, Ph.D.

Delphi and ASP

Getting Started with Active Server Pages

In brief, Active Server Pages (ASP) are text files that contain standard HTML commands that are processed by a Web browser, and scripting commands that are executed on a Web server. These scripting commands can be written in VBScript, JScript (the Internet Explorer equivalent of Netscape's JavaScript), or any other language for which a valid scripting engine is installed, e.g. Perl and Python. Using these scripting commands, you can introduce basic programming operations into your HTML, such as performing conditional execution and expression evaluation, as well as invoking the features of COM servers installed on the server. In this article, we'll take a look at using these scripting commands to achieve the results you want.

When an HTTP request for an ASP document is received by an ASP-enabled Web server, such as Microsoft's Internet Information Server (IIS), the server processes the scripting commands within that file. The output generated by the scripting language is combined with the standard HTML in the ASP to produce the HTTP response returned to the Web browser. In other words, the server doesn't return the ASP document to the browser. Instead, it returns static HTML, plus the output from the processing of the scripting commands. In most cases, the returned data is pure HTML, although it can potentially contain any valid MIME (Multi-Purpose Internet Mail Extensions) content type.

There are numerous benefits to using ASP:

- The user can't readily access your code. Unlike with browser-side JavaScript or VBScript, in which the scripting instructions are delivered intact to the browser, ASP commands aren't sent to the Web browser.
- ActiveX objects accessed using ASP — when Microsoft Transaction Server (MTS) isn't being used — are loaded into the process space of the server, which reduces overhead when compared with CGI (common gateway interface) server extensions. An ASP object can also be processed in a separate process space, reducing the likelihood that a crashing ASP object will bring down your Web server.
- ActiveX objects used by IIS can be installed into MTS, providing activation on demand, transactions, resource sharing, crash protection, and additional security.
- ASP code is often easier to write than other CGI programs, in that it doesn't require compilation or the installation of a secondary processor (although when you create a COM server designed to be accessed by an ASP, the COM server requires compilation).

As noted already, ASP can only be used with an ASP-enabled Web server. Initially, ASP support was introduced in Microsoft's Internet Information Server (IIS). However, any Windows-based server can potentially support ASP. This is crucial in that ASP servers use COM technology, which is supported almost exclusively by the Windows operating system. (Although some non-Windows implementations of COM exist, these are quite rare.)

Before continuing, it's worth mentioning that JSP (Java Server Pages) is a technology similar to ASP. The primary difference is that JSP-enabled servers aren't limited to Windows-based servers. Any Web server that runs on a Java 2-supporting operating system (JDK 1.2) can be a potential JSP-enabled server.

Using ASP Commands

In addition to basic HTML, ASP contains four types of commands: primary scripting commands, output directives, processing directives, and include statements.

With the exception of include statements, commands contained within an ASP document are enclosed within <% %> delimiters. As mentioned earlier, these commands can consist of VBScript, JScript, or any other scripting language for which a valid scripting engine has been installed on the Web server. The language used for most of the remainder of this article is VBScript.

Primary scripting commands contain the basic syntactic elements of the scripting language being used. For example, primary scripting commands may include control structures, expression evaluation, function invocation, variable declarations, and so on. HTML commands can be interspersed between primary scripting commands to control the HTML content that is returned to the Web browser. For example, consider the following segment:

```
<%
If Time >= #12:00:00 AM# And Time < #12:00:00 PM# Then
%>
Good Morning!
<% Else %>
Hello!
<% End If %>
```

Here, the plain HTML text `Good Morning!` appears between the `<%If%>` and the `<%Else%>` commands. Alternatively, the *Write* method of the built-in *Response* object can be used to write HTML from within a primary scripting command. The *Response* object is one of six available built-in objects you can use in your ASP. Built-in objects are discussed in greater detail later in this section. The following is an example of ASP commands that make use of the *Response.Write* method:

```
<%
If Time >= #12:00:00 AM# And Time < #12:00:00 PM# Then
  Response.Write "Good Morning!"
Else
  Response.Write "Hello!"
End If
%>
```

In this example, the entire code segment is a primary scripting command. The result is that one of two strings is written to the HTML file being created by the Web server based on the evaluation of the `<%If%>` statement. In other words, the result of this segment will either be `Good Morning!` or `Hello!`. None of the other actual text of the primary scripting command is delivered to the browser.

Output directives. Output directives are a special case of primary scripting commands. As with the *Response.Write* technique, an output directive inserts text into the HTML stream that will be returned to the Web browser. Output directives use the following syntax:

```
<%= expression %>
```

The value of expression can be a variable, constant, formula, property, method, or function. For example, the following line inserts a string indicating the current time, based on the internal clock on the Web server (because the Web server can be at any location in the world, this isn't necessarily the same time as in the time zone of the Web browser!):

```
The current time on this server is: <%=Now%>
```

Processing directives. Processing directives provide instructions to the ASP processing engine and typically appear at the beginning of an ASP page. Processing directives use the following syntax:

```
<% @ keyword=value%>
```

The keyword must be a reserved word recognized by the ASP engine and it must be separated from the @ sign by at least one space. The keyword is followed by an equals sign (=) and a value. Furthermore, multiple keywords can appear in a single processing directive, and no spaces can appear on either side of the equals sign. The following is an example of a processing directive that tells the Web server that the VBScript language is being used (the default):

```
<% @ LANGUAGE=VBScript %>
```

When using VBScript or JScript, the ASP processing engine removes white space. When you need to include a blank space in HTML, use the HTML non-breaking white space character, &nbsp; see an HTML reference for additional special characters.

VBScript Comments and Variables

In VBScript, comments can appear within primary scripting commands, with the exception of output commands. Comments are defined by anything to the right of an apostrophe (with the exception of the end delimiter):

```
<%
If Time >= #12:00:00 AM# And Time < #12:00:00 PM# Then
  Response.Write "Good Morning!"  ' Write morning greeting.
Else
  Response.Write "Hello!"  ' It must be later in the day.
End If
%>
```

As you can imagine, comment notation depends on the scripting language. For example, while VBScript uses the apostrophe, JScript makes use of the // characters to denote a comment, similar to Java, Delphi, and C++.

Although variables don't need to be declared in VBScript, they should be declared for good programming practice. Once declared, or assigned in those cases in which they aren't declared, variables are accessible to the remaining commands in that ASP page. For example, the following statement declares a variable named *UserName*, and sets its value to that of a value passed in the HTTP query string. The following variable can be used in any expression that appears in any commands later in this page:

```
<%
Dim UserName
Set UserName = Response.QueryString("Name")
%>
```

There are two special classes of variables that are visible to more than one ASP page. These are *Session* and *Application* variables. *Session* variables are available to all pages accessed by a particular user within a session. By comparison, *Application* variables are shared by all ASP pages in an ASP application. These variables are stored using the *Session* and *Application* built-in objects, respectfully. These objects are described in greater detail later in this article.

You access *Session* and *Application* variables by passing the name of the variable to the *Session* or *Application* object. For example, the following code stores the session start time in a *Session* variable:

```
<% Session("SessionStart") = Now %>
```

All pages on the site accessed by a specific user can then read this value using the following statement:

```
You began your session at <% = Session("SessionStart") %>
```

VBScript doesn't permit the declaration of constants. Constants are defined using type libraries.

Include Files

Include files are similar to Delphi's Web Broker *TPageProducer* in that they permit you to insert segments of HTML into your page at specific points. For example, if you have a segment of HTML that should appear at the top of every page, you can create it once and embed it within each ASP page by using the include directive before any other simple HTML text or commands on the page.

The include statement has the following syntax:

```
<!--#include VIRTUAL|FILE="dirname/inc.htm"-->
```

If the keyword VIRTUAL is used, the dirname part is a virtual directory (defined through server configuration). If FILE is used, the directory can either be an absolute or a relative directory. For example, if you have created a virtual directory under IIS named Chunks, you can use the following include statement to include the header.htm HTML in your ASP page:

```
<!--#include VIRTUAL="Chunks/header.htm"-->
```

In addition to including static HTML, include files can also reference ASP pages or ASP segments. However, note that ASP segments referenced by include directives are processed on the server before the referencing ASP page is processed. This processing is unconditional. That is, all included ASP pages are processed — even those where the include statement is referenced within an <%IF%> statement — before the first primary command of the referencing ASP page is processed.

Using Objects

ASP objects are Automation servers, and therefore can be either in-process servers or out-of-process servers. There are two general classes of objects that you can use from an ASP: built-in objects and custom objects.

The *Server* and *Application* objects referenced previously are examples of built-in objects. These objects are available to all ASP pages. These built-in objects are automatically created for you by the server, and can be referenced within any primary commands. By comparison, you must explicitly create custom objects before you can access them. You create custom objects by calling the *CreateObject* method of the *Server* built-in object, passing to it the PROGID of the registered object. For example, if you have registered an Automation server with the PROGID of *Project1.Text*, you can create an instance of it, and assign this instance to the variable *DelphiASPObj* using the following command:

```
<%
Set DelphiASPObj = Server.CreateObject("Project1.Test")
%>
```

You can also use the HTML <OBJECT> tag to create an instance of an object. When doing so, however, you must include the RUNAT directive with the *Server* option to ensure that this object runs on the server, and not on the browser's machine:

```
<OBJECT RUNAT=Server ID=DelphiASPObj
  PROGID="Project1.Test"></OBJECT>
```

Java objects. Rather than using COM objects, you can use Java objects in your ASP commands, as long as your Web server supports Java. The following example demonstrates how to create an instance of the *java.lang.Integer* class:

```
<%
Dim date Set intobj = GetObject("java:java.lang.Integer ")
%>
```

Once you have a reference to the Java object, you can call its methods. For example, the following example calls the *parseInt* method of the *Integer* class:

```
<% = intobj.parseInt(somestring) %>
```

Although the availability of Java objects within ASP pages provides you with some flexibility, Java objects typically can't access the built-in objects, nor can they be part of an MTS transaction. Consequently, their use is usually reserved for special operations not normally available through the scripting language of COM objects.

Using methods. You call an object's methods using dot notation to invoke the qualified method name. If the method requires one or more parameters, then follow the method name using the syntax of your scripting language. For example, to invoke the built-in *Response* object's *Write* method, you use a statement similar to the following example:

```
<% Response.Write "Text to return to the browser" %>
```

Using properties. You reference an object's property by using dot notation to reference the qualified property. You write to the property by placing the property reference on the left side of an assignment statement. You read the property by including the qualified property name in an expression. Because properties in COM make use of accessor methods, reading and writing properties of ASP objects results in the execution of the defined read and write methods, respectively. From these methods, you can invoke custom code, or even reference both built-in and custom ASP objects.

Built-in objects. ASP pages have access to a number of objects created automatically by the ASP-enabled server. These objects can be referenced by your server-side VBScript, permitting you to get information about the environment, as well as to control the behavior of your ASP. The following built-in objects are available to all ASP pages: *Application*, *ObjectContext*, *Request*, *Response*, *Server*, and *Session*.

The *Application* object permits you to store application-wide data. An ASP application is a set of related ASP pages under a common directory structure. The application starts the first time a page of the application is loaded, and shuts down when the server is shut down.

The *ObjectContext* object is used to start, commit, abort, and complete transactions.

The *Request* and *Response* objects permit you to get information about the HTTP request and to control the HTML response. For example, the *Request* object permits you to read query strings, cookies, binary data, client certificates, and server variables. The *Response* object, by comparison, can be used to write cookies, control page caching (affecting caching servers), and write the response, among other operations.

The *Server* object is used primarily to create new ActiveX or Java objects. It can also be used to control URL mapping and encoding and HTML encoding.

The *Session* object represents one or more hits on pages in a given ASP application by the same user. To use a session object, the user's browser must be set to accept cookies to be sent back to the same domain. One of the most powerful aspects of the *Session* object is that it permits you to track information as a user navigates from one page to another within your ASP application.

If you must track sessions without using cookies, you can use Cookie Munger. Cookie Munger is an IIS filter that executes for every page request, producing an overall performance penalty as an unwanted side effect. Cookie Munger will detect whether the browser will accept cookies. If the browser doesn't accept cookies, the munger will generate a session ID for that browser. Furthermore, when a page is being sent back to a browser that doesn't accept cookies, Cookie Munger adds query string information to every URL that references back to the ASP application. The Microsoft Cookie Munger can be found in the IIS Resource Kit, and more information is available at http://msdn.microsoft.com/workshop/server/toolbox/cookie.asp.

Custom ASP objects. Even without custom ASP objects, it's clear that ASPs can provide you with the power and flexibility to create more intelligent Web content. However, one of the more powerful aspects of ASPs is that you can create custom ActiveX servers for use with them. Specifically, you can create ActiveX servers that are invoked by a Web server at run time. Those ActiveX servers can read information about the HTTP request, as well as execute custom code to provide the necessary response. ActiveX servers used in this context are referred to as ASP objects for short.

Since the release of Delphi 3, Delphi has made the creation of COM servers almost effortless. With Delphi 5, this convenience has been extended to ASP objects. The Active Server Object Wizard, shown in Figure 1, appears on the ActiveX page of the New Items dialog box. Using this wizard, you can create an ASP object that can be registered on your Web server (in that machine's Windows registry), as well as a sample HTML page that can be used to invoke your custom ASP page.

As mentioned earlier in this section, custom ASP objects can be in-process servers or out-of-process servers. The following example makes use of in-process ASP objects. These objects can be loaded in the process space of the Web server, or installed and invoked from MTS.

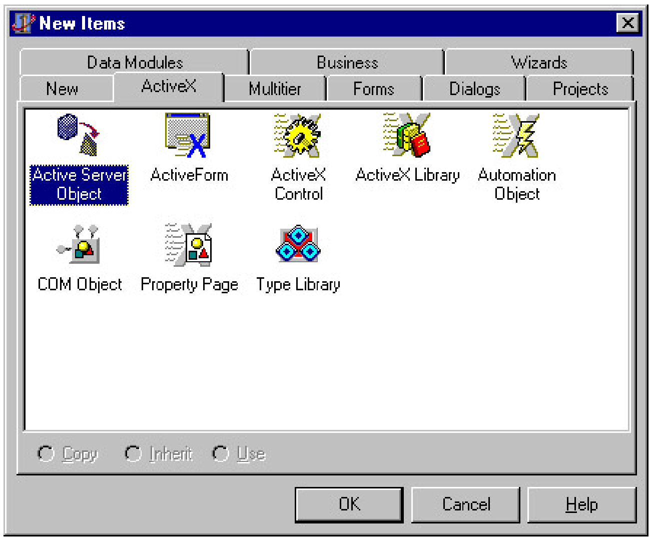


Figure 1: The Active Server Object Wizard is accessible from the New Items dialog box.

Creating a Custom ASP Example

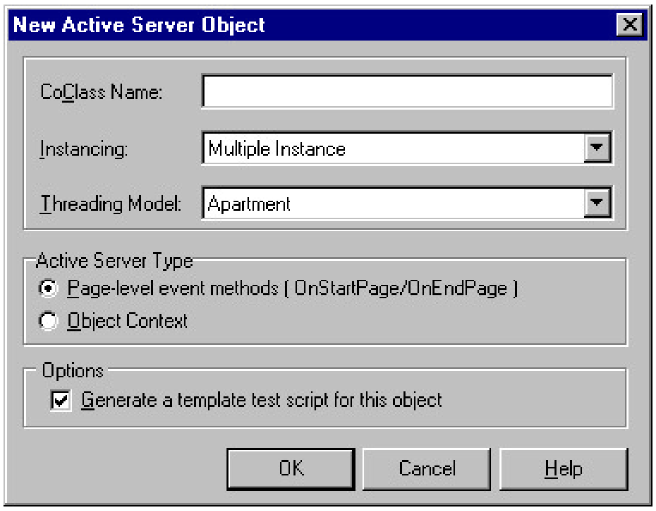
Use the following steps to create a custom ASP object for use with IIS:

1) Select File | Close All.
2) Select File | New to display the Object Repository. Select the ActiveX tab, then double-click the ActiveX Library Wizard. This wizard creates a new DLL-based project that exports the essential four functions for in-process COM server activation.
3) Select File | New again. Then, select the ActiveX tab and double-click the Active Server Object Wizard. Delphi responds by displaying the New Active Server Object dialog box (see Figure 2).
4) At CoClass Name, enter `DelphiASP`. If you're using IIS version 3.0 or 4.0, you must set the Active Server Type to Page-level event methods. If you're using IIS version 5.0, select Object Context. Leave Generate a template test script for this object enabled. Click OK when you're done.

At this point, creating your ActiveX server object is like creating any other type of COM server. You use the Type Library editor to declare methods and properties. From within the implementation of your methods, you write your code logic. What is somewhat different from other types of COM servers is that the interfaces that enable ActiveX server objects expose a number of interface references that provide you with access to the built-in server objects. For example, your server inherits a *Request* property, which gives you access to the built-in *Request* object. Similarly, you have access to the *Response* property, which you use to invoke methods of the built-in *Response* object.

The following steps demonstrate how to implement a method on the server. This particular example is being kept fairly simple.

1) Using the Type Library editor, add a new method named *SayHello*. Don't define any parameters for this method (see Figure 3).
2) Click the Refresh Implementation button to generate the *SayHello* method stub in the *DelphiASP* CoClass.
3) Implement the *DelphiASP.SayHello* method. The example in Figure 4 demonstrates the use of several *Response.Write* methods, using the *ServerVariables* interface, as well as accessing the *Request* object.
4) Save the project as DelphiASP.DPR. (Save the CoClass unit using any name you want. In the ASPDemo project, it's named coclassu.pas.)
5) Compile the project, and then select Run | Register ActiveX to register this ASP object with the Windows registry. (All files and projects are available for download; see end of article for details.)



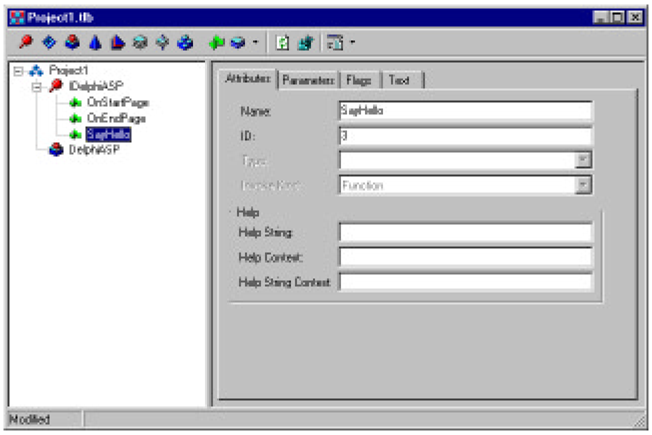
CoClass Name:
Instancing: Multiple Instance
Threading Model: Apartment
Active Server Type
(●) Page-level event methods (OnStartPage/OnEndPage)
( ) Object Context
Options
[✓] Generate a template test script for this object

Figure 2: The New Active Server Object dialog box.

Calling the Custom ASP Object

You now must update the text of the ASP page generated by Delphi. Originally this page looked like the code shown in Figure 5.

Change the generated code to look like that shown in Figure 6.


Project1.tlb

Name: SayHello
ID: 3
Type:
Invoke Kind: Function
Help String:
Help Context:
Help String Context:

Modified

Figure 3: Adding the *SayHello* method in the Type Library editor.

```
procedure TDelphiASP.SayHello;
var
  i: Integer;
begin
  with Self.Response do begin
    Write('This is the response '+
      'from the Delphi ASP object. <P>Here is a big ' +
      '<H2><CENTER><I>Hello !!</I></CENTER></H2><P>' +
      'This request has come from a ');
    Write(Request.ServerVariables.Get_Item(
      'HTTP_USER_AGENT'));
    Write(' agent at address ');
    Write(Request.ServerVariables.Get_Item('REMOTE_HOST'));
    Write('.<BR> Furthermore, the last page you '+
      'visited was (none if blank): ');
    Write(Request.ServerVariables.Get_Item(
      'HTTP_REFERER'));
    Write('<P>The request method was ');
    Write(Request.ServerVariables.Get_Item(
      'REQUEST_METHOD'));
    Write('<P>There were ');
    Write(IntToStr(Request.QueryString.Count));
    Write(' items in the query string.');
    if Request.QueryString.Count <> 0 then
      begin
        Write('<P>The query string is :');
        Write(Request.QueryString);
      end;
    if Request.Form.Count <> 0 then
      begin
        Write('<P>There were ');
        Write(IntToStr(Request.Form.Count));
        Write(' items sent by an HTML FORM.');
        Write('<P>These are ');
        for i := 1 to Request.Form.Count do begin
          Write('<BR>');
          Write(Request.Form.Key[i]);
          Write(' : ');
          Write(Request.Form.Item[i]);
        end;
      end;
    if (Request.QueryString.Count = 0) and
       (Request.Form.Count = 0) then
     Write('<P>There was no data sent with this request.');
  end;
end;
```

Figure 4: Demonstrating the use of several *Response.Write* methods.

In this modified ASP page text, the *CreateObject* call has been updated to refer to the ProgID of the newly created ASP object. Also, the call to invoke the *SayHello* method of this object has been added.

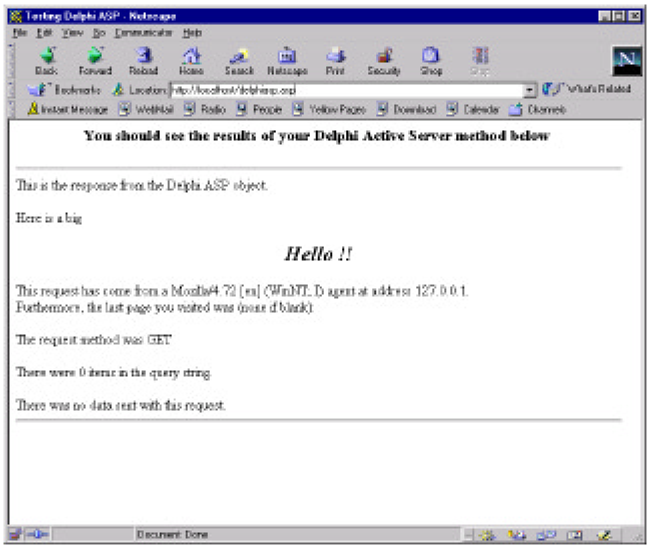
Now save the .asp file to the root directory of your Web server. Make sure you keep the file extension as .asp. Next, using the Ctrl O (**File | Open**) selection from your Web browser, type `http://localhost/delphiasp.asp`. (This assumes you're testing this application using a local Web server. If you're using a Web server on another machine, replace *localhost* with either the domain name, the machine name, or the IP address of your Web server.) After a moment, the page should load. Figure 7 shows what this page looks like in Netscape Communicator.

```
<HTML>
<BODY>
<TITLE> Testing Delphi ASP </TITLE>
<CENTER>
<H3> You should see the results of your Delphi Active
 Server method below </H3>
</CENTER>
<HR>
<%
Set DelphiASPObj=Server.CreateObject("Project1.DelphiASP")
DelphiASPObj.{ Insert Method name here. }
%>
<HR>
</BODY>
</HTML>
```

Figure 5: Original ASP page.

```
<HTML>
<BODY>
<TITLE> Testing Delphi ASP </TITLE>
<CENTER>
<H3> You should see the results of your Delphi Active
 Server method below </H3>
</CENTER>
<HR>
<% Set DelphiASPObj = Server.CreateObject(
  "DelphiASP.DelphiASP") DelphiASPObj.SayHello %>
<HR>
</BODY>
</HTML>
```

Figure 6: Modified ASP page.


Testing Delphi ASP - Netscape

Location: http://localhost/delphiasp.asp

You should see the results of your Delphi Active Server method below

This is the response from the Delphi ASP object.

Here is a big

Hello !!

This request has come from a Mozilla/4.72 [en] (WinNT; I) agent at address 127.0.0.1.
Furthermore, the last page you visited was (none if blank):

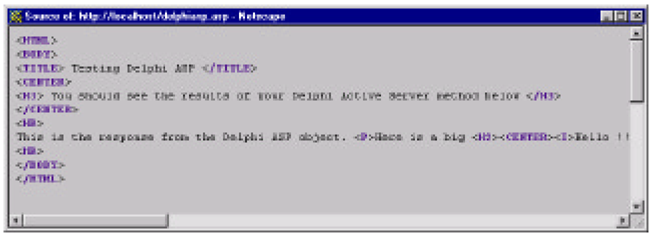
The request method was GET

There were 0 items in the query string.

There was no data sent with this request.

Document: Done

Figure 7: The updated ASP page as seen in Netscape Communicator.

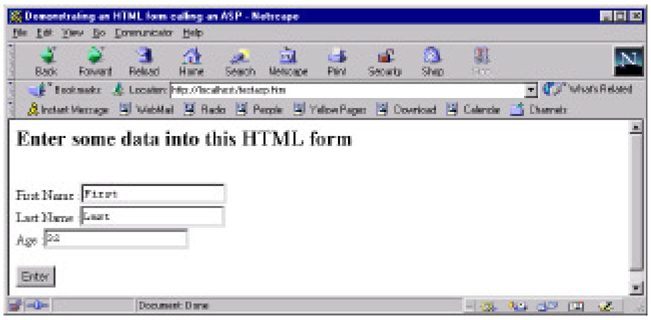


```
<HTML>
<BODY>
<TITLE> Testing Delphi ASP </TITLE>
<CENTER>
<H3> You should see the results of your Delphi Active Server method below </H3>
</CENTER>
<HR>
This is the response from the Delphi ASP object. <P>Here is a big <H1><CENTER><I>Hello !!
<HR>
</BODY>
</HTML>
```

Figure 8: The HTML generated by ASP.

```
<HTML>
<TITLE>Demonstrating an HTML form calling an ASP</TITLE>
<BODY>
<H2>Enter some data into this HTML form</h2>
<FORM  Method=GET Name=Form Action=delphiasp.asp>
<BR>First Name :<INPUT TYPE="text" NAME="firstname">
<BR>Last Name :<INPUT TYPE="text" NAME="lastname">
<BR>Age :<INPUT TYPE="text" NAME="age">
<INPUT TYPE="hidden" NAME="userstatus" VALUE= "new">
<P><INPUT TYPE="submit"  VALUE="Enter">
</FORM>
</BODY>
</HTML>
```

Figure 9: An HTML form calling an ASP page.

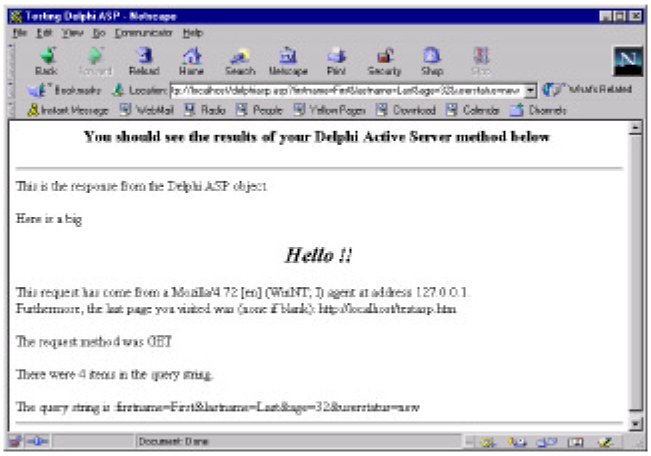


Enter some data into this HTML form

First Name: First
Last Name: Last
Age: 32

Enter

Figure 10: The HTML text from Figure 9 displayed in a browser.



You should see the results of your Delphi Active Server method below

This is the response from the Delphi ASP object.

Here is a big

Hello !!

This request has come from a Mozilla/4.72 [en] (WinNT; I) agent at address 127.0.0.1.
Furthermore, the last page you visited was (none if blank): http://localhost/testasp.htm

The request method was GET

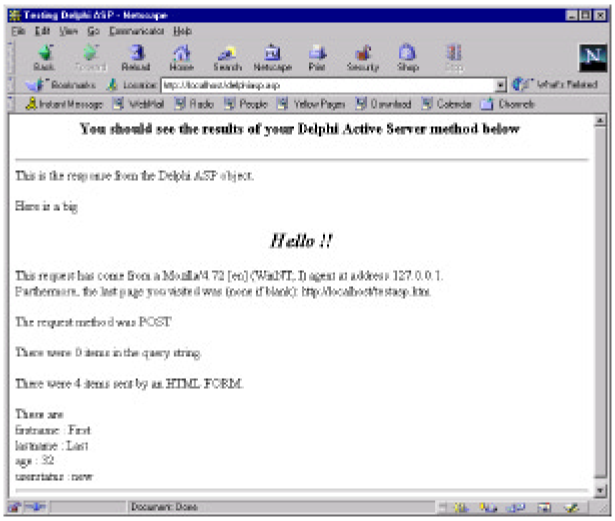
There were 4 items in the query string.

The query string is :firstname=First&lastname=Last&age=32&userstatus=new

Figure 11: The resulting HTML after invoking your ASP object.

Because this page was displayed using the Open Page feature, there was no HTTP referrer. However, if you loaded this page by clicking an anchor tag link (<A>), or submitting an HTML form with either a GET or POST action, the URL of that linking page would be displayed following the "last page you visited" text.

If you now select **View | Page Source**, you will see the HTML generated by the ASP (see Figure 8). Notice that what you see here is plain HTML. None of the ASP commands are visible, because the server replaced them with the HTML text generated by the ASP object.



You should see the results of your Delphi Active Server method below

This is the response from the Delphi ASP object.

Here is a big

Hello !!

This request has come from a Mozilla/4.72 [en] (WinNT; I) agent at address 127.0.0.1.
Furthermore, the last page you visited was (none if blank): http://localhost/testasp.htm

The request method was POST

There were 0 items in the query string.

There were 4 items sent by an HTML FORM.

There are
firstname : First
lastname : Last
age : 32
userstatus : new

Figure 12: HTML opting for the GET action rather than the POST action.

Calling Your ASP from an HTML Form

As mentioned earlier, if you had displayed your ASP-generated HTML by clicking on a link, the HTTP_REFERER server variable would contain the URL of the linking page. Likewise, if this link was generated by either a GET or POST action from an HTML form, the ASP would display the data sent by the form.

This effect is demonstrated using the HTML page defined by the text in Figure 9, which is found in the TESTASP.HTM file located along with the sample code for this article.

If you save this page to the same directory in which you have placed the ASP page, then load it into your browser and enter data into the text input fields, your browser will look something like that shown in Figure 10.

If you click the **Enter** button on the HTML form to invoke your ASP object, the resulting HTML downloaded to your browser may look similar to that shown in Figure 11.

Alternatively, if your HTML form used the GET action (the default) instead of POST, the resulting page may look something like that shown in Figure 12.

Conclusion

In this article, we've seen a number of different ways you can benefit from the use of ASP in Web development. And when you add ASP's ability to utilize Delphi's COM capabilities, Web development becomes all that much more robust. ∆

The files referenced in this article are available on the Delphi Informant Magazine Complete Works CD located in INFORM\00\JUL\DI200007CJ.

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 Bill Todd

Wise for Windows Installer 2.0

The Future of Windows Installation

Windows Installer is Microsoft's new technology for controlling the installation of software and operating system updates on systems running Windows 2000 and Windows Millennium. Adding installation support to the operating system is part of Microsoft's effort to improve stability. By gaining control of the installation process, Microsoft can ensure that installing application software cannot damage the operating system and, hopefully, other applications. Wise Solutions' Wise for Windows Installer 2.0 lets you quickly and easily build installations that use Windows Installer.

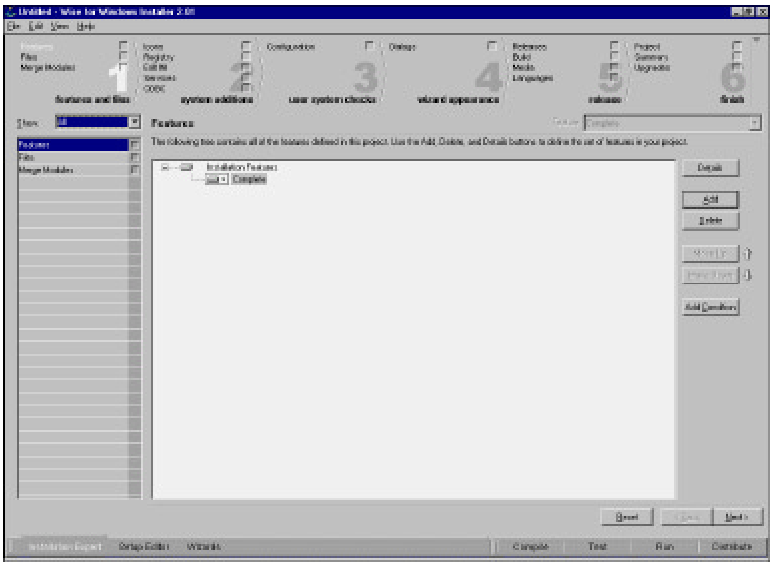
Windows Installer brings a number of new concepts and terms to the world of software installation. If you are already familiar with Windows Installer, you can skip this section. In introducing a new installation technology for its new family of operating systems, Microsoft did not want to force developers to create one installation using traditional technology for Windows 95, 98, and NT 4, and another using Windows Installer for Windows 2000 and Millennium. To solve this problem, Microsoft has created versions of Windows Installer that will run on Windows 95, 98, and NT 4.

To install software using Windows Installer, you must create a special relational database that contains all of the information Windows Installer needs to install your software. The database is contained in a single file with an .msi extension. Wise for Windows Installer lets you build this database.

Learning to use Wise for Windows Installer, or any other Windows Installer tool, is much easier if you have a basic understanding of Windows Installer concepts and terminology. You will find the "Roadmap to Windows Installer Documentation" at http://msdn.microsoft.com/library/psdk/msi/leglivt_0002.htm. From this page, you can access all of the information Microsoft has published about Windows Installer. You may also want to download the Windows Installer SDK (about 6MB) from http://download.microsoft.com/msdownload/platformsdk/i386/InstallerSamples/IntelSDK.msi. Even if you do not plan to use the SDK to develop your own installation software or interact with the Windows Installer through COM, the SDK Help file is a valuable resource for learning about Windows Installer.

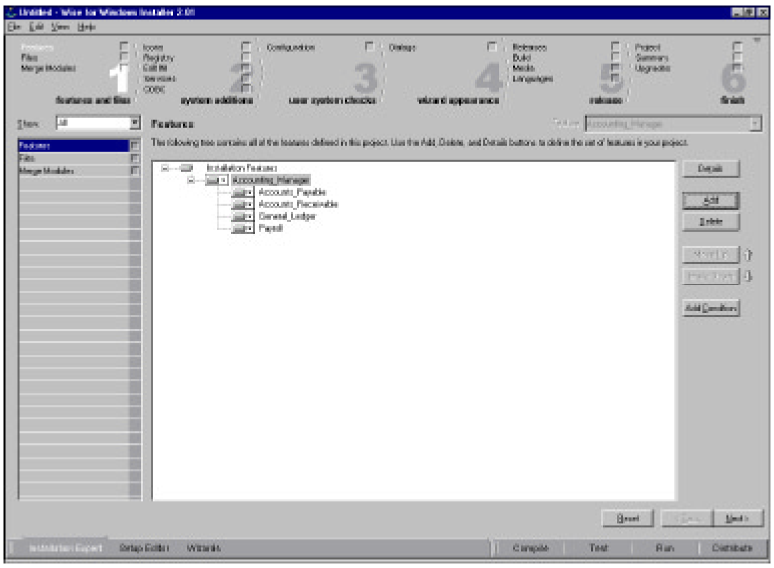
Windows Installer offers a host of new features, such as self-healing applications. When you launch a self-healing application, it will automatically detect missing or damaged files, and automatically reinstall the missing files. An application can ask Windows Installer if a specific application feature is installed, and alter its appearance and behavior based on the answer. In large organizations, administrators can advertise applications that are available to users. These applications will appear on the user's **Start** menu, just as if they were installed, and Windows Installer will automatically install the application the first time the user opens it.

Windows Installer also lets you install features of your application on demand. If a user chooses a feature that isn't installed, one of two things will happen, depending on whether the application was installed from removable media or a network location. If the original installation was from a network location that



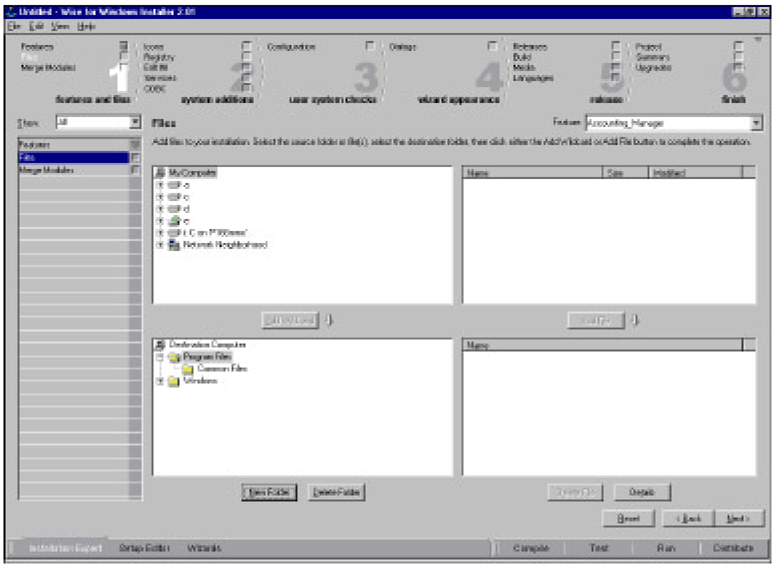
The following tree contains all of the features defined in this project. Use the Add, Delete, and Details buttons to define the set of features in your project.

Figure 1: The Wise for Windows Installer 2.0 installation expert.



The following tree contains all of the features defined in this project. Use the Add, Delete, and Details buttons to define the set of features in your project.

Figure 2: A hierarchy of features.



Add files to your installation. Select the source folder or file(s), select the destination folder, then click either the Add Wildcard or Add File button to complete the operation.

Figure 3: Adding files to your installation.

is still accessible, Windows Installer will automatically install the requested feature. If the original installation was from removable media, Windows Installer will prompt the user to insert the media, then install the feature.

Using the Installation Expert

If you have used version 7 or higher of any of the Wise installation products, Wise for Windows Installer will look very familiar. When you start Wise for Windows Installer, you will see the installation expert screen, shown in Figure 1.

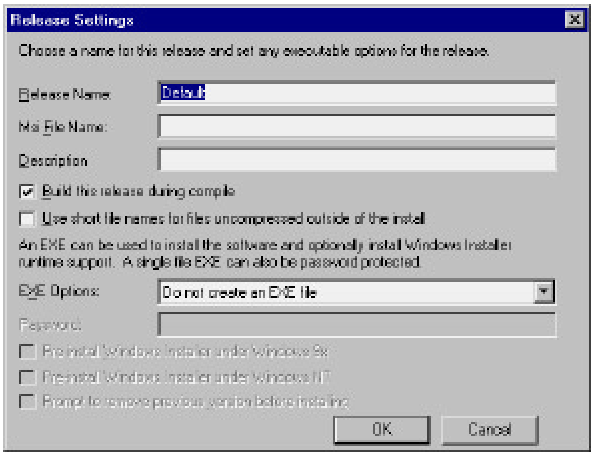
The installation expert separates the process of creating an installation into six steps, most of which are divided into one or more subtasks. You can move from step to step by clicking the step buttons at the top of the screen. You can also click a task in any button at any time to jump directly to that task. To move through the process in order, use the **Next** and **Back** buttons at the bottom right of the screen.

Windows Installer installs products. A product may consist of one or more features. A typical example might be an accounting package that is separated into features, such as general ledger, accounts receivable, accounts payable, payroll, and so on. You must have at least one feature, so Wise for Windows Installer automatically creates a feature named Complete. You cannot delete Complete until you have added at least one other feature. Assume you are going to install an accounting system, and that all of the accounting modules depend on a module named Accounting Manager that must always be installed.

To create this structure, click **Features** in step 1 of the installation expert, then click **Installation Features** in the tree view. Click the **Add** button to display the Feature Details dialog box, and enter `Accounting Manager` as the name. Next, select **Complete** in the tree view, and click the **Delete** button to remove it. Because the Accounting Manager must be installed for any of the other modules to work, you can make the other modules dependent on the Accounting Manager by adding them under the Accounting Manager feature. Just click on **Accounting Manager** in the tree view, click the **Add** button, and enter the name of the new feature. Repeat this for each feature you want to add until you have the structure shown in Figure 2. By making features dependent on each other, you can ensure that users always install everything necessary to make the features they select work.

You can organize features to provide uses with the option of doing a Typical, Complete, or Custom installation, although the mechanism for doing this was not immediately obvious. You must set the **Level** property to *Custom* in the Feature Details dialog box, then set the **Custom Value** to three or less to include the feature in a typical install, or 1,000 or less to include the feature in a complete install. Wise plans to improve this process in a future version.

Once you've defined your features, you must define the files that must be installed for each feature. Choose **Files** in step 1 to display the file selection screen, shown in Figure 3. When the Files display appears, the **Features** drop-down list, just



Release Settings

Choose a name for this release and set any executable options for the release.

Release Name: Default

Msi File Name:

Description

[✓] Build this release during compile

[ ] Use short file names for files uncompressed outside of the install

An EXE can be used to install the software and optionally install Windows Installer runtime support. A single file EXE can also be password protected.

EXE Options: Do not create an EXE file

Password:

[ ] Pre-install Windows Installer under Windows 9x

[ ] Pre-install Windows Installer under Windows NT

[ ] Prompt to remove previous version before installing

OK Cancel

Figure 4: The Release Settings dialog box.

below the buttons for steps 5 and 6, is enabled. All of the files and folders you add to your installation apply to the selected feature; that is, they will be installed when that feature is installed.

The lower-left pane in Figure 3 shows the directory structure on the target computer. To add a new subdirectory, select its parent in the directory tree and click the **New Folder** button. The upper-left pane shows the directory structure on your computer. To add files to a folder, locate the files on your computer, select them, and click the **Add File** button to add them to the selected folder in the destination computer pane.

If you need to add an entire directory or directory structure, select the parent directory on the destination computer, select the directory you want to add to the installation on your computer, and click the **Add Wildcards** button. You can specify both include and exclude filters to include all files that match the template you enter, or exclude files that match a template you define. For example, you could set the include filter to *.EXE;*.DLL;*.OCX to include all of the executable, DLL, and ActiveX control files in the directory. You can also include subdirectories, and tell Wise for Windows Installer to automatically update the installation each time it's compiled to reflect the files currently in the source directory.

The last option in step 1 is **Merge Modules**. A merge module is a pre-compiled installation designed to allow third parties to give you a way to install their products with your own. Hopefully by the time you read this there will be available a merge module for the BDE. To add a new merge module to the feature you have selected, click the **Add New** button to display the Add Merge Module Wizard. You can choose a merge module from the list of modules that comes with Wise for Windows Installer, or click the **Browse** button to locate a module on your hard disk. If the feature you are working with needs a merge module that you've already added to another feature, click the **Add Existing** button and choose the module you need.

Moving to step 2 gives you options to install icons; add, delete, or change registry settings; create, delete, or change .ini file settings; install Windows services; or add ODBC drivers or data sources. These options are almost identical to the corresponding options in Wise InstallMaster 8.0, another installation product from Wise Solutions. The only difference is that you set each of these options for each feature in your installation.

Step 3 lets you check the user's system configuration for a minimum version of Windows or Windows NT, a minimum screen resolution, and a minimum color depth. If the configuration on the destination computer doesn't meet the minimum requirements, the warning message you enter will be displayed.

Step 4 consists of a single choice, **Dialogs**. Here, you set which dialog boxes will be displayed when a user runs your installation. If you choose to include the License or Read Me dialog boxes, you can enter the text of your license agreement or read me file in .rtf format, or import the text from an existing .rtf file.

In step 5, you define the releases you want to build. A release is some combination of features, media, EXE options, and languages. By creating multiple releases, you can have a single installation that creates your diskette release and your CD-ROM release for both the standard and professional versions of your product in each different language that you need.

There must be at least one release, so Wise for Windows Installer automatically creates one, named Default. You can click the **Details** button to change the name or other properties of the Default release, and click the **Add** button to add a new release. Figure 4 shows the Release Settings dialog box used to add or edit a release.

The key setting in this dialog box is **EXE Options**. If you know the destination computer is running Windows 2000, or that it already has Windows Installer installed, you can safely choose the **Do not create an EXE file** option. In this case, only the MSI database file required by Windows Installer will be created when you compile your installation. If the target computer is running Windows 95, 98, or NT 4, and you cannot be sure that Windows Installer has already been installed, you will want to choose one of the options that creates an installation EXE. When you create an installation EXE, launching the EXE on the destination computer will first install Windows Installer, then use Windows Installer to install your application. Creating an EXE adds about three megabytes to the size of your installation.

There are two EXE options. The first is **Single File Executable**, and is suitable for a single file installation that will fit on a CD-ROM, or will be downloaded or installed from a network. If you need to create installation diskettes or CDs, choose **Executable That Launches External MSI**.

Moving to the Build page lets you set the properties, summary items, and features to be included in the selected release. Properties are variables supported by Windows Installer, such as **ProductName**, **ProductVersion**, and **DiskPrompt**. If you want to know the purpose and legal values for all of the property variables, you will need the Windows Installer SDK Help file or other Windows Installer documentation. Summary information consists of information that can be displayed by right-clicking the MSI file. Examples are Author, Comments, Keywords, and Minimum Installer Version.

The Build page also shows the feature tree with a checkbox next to each feature. Simply check the features you want included in this release. The Media page, in step 5, allows you to define the media settings for the selected release. You can opt for a single installation file with all installation files compressed into the MSI database, or you can choose to have files compressed into CAB files outside the MSI. This is the option you must choose if your installation will span multiple disks. You can use any type of media for your installation by setting the media size and the cluster size.

If your installation will span multiple disks, you must create and enter the path to a subdirectory to hold the contents of each disk.

Because the installation will consist of the MSI file and multiple CAB files, each diskette may hold more than one file. Using subdirectories for each diskette is the only way to keep the files organized by the diskette on which they belong. The final option in step 5 is to choose whether you want the installation to run in French, Italian, German, Portuguese, or Spanish, instead of English.

The sixth — and final — step in creating your installation starts by prompting you for project information. The most important item on this page is the Product Code. The Product Code is a GUID used by Windows Installer to determine if this product is already installed on the destination computer. This code must be different for different languages and versions. The Summary page provides another opportunity to enter the summary information described earlier in step 5. The Upgrades page allows you to identify features of an existing installation to remove as part of this installation. The final three options are Windows 2000, Status MIF, and Code Signing. The Windows 2000 page lets you configure the options available to the user in the enhanced Add/Remove Programs dialog box in Windows 2000. On the Status MIF page, you can configure your installation to be run under Microsoft Systems Management Server. The Code Signing page allows you to create a code-signed single file installation for Internet distribution.

Using the Setup Editor

The Setup Editor, shown in Figure 5, provides a completely different view of your installation, as well as the ability to edit the Windows Installer database tables directly.

The Setup Editor screen is divided into three panes. The left pane contains six tabs that correspond to the types of information you can work with. A different tree view appears for each tab. The upper-right pane displays the detailed values for the selected item in the left pane. For example, if you select the Product tab, the left pane contains Launch Conditions, Properties, and Summary. If you click Properties, all of the properties and their values appear in the upper-right pane. The lower-right pane contains help about the selected item in the left pane.

Properties and Summary information have already been described. Launch Conditions lets you create conditions that must be true for the installation to run. For example, you could require that the user be logged on as Administrator to install your application. Wise for Windows Installer tries to make creating launch conditions easy by providing a Condition Builder. The Condition Builder lets you build conditional expressions by making point-and-click choices from lists.

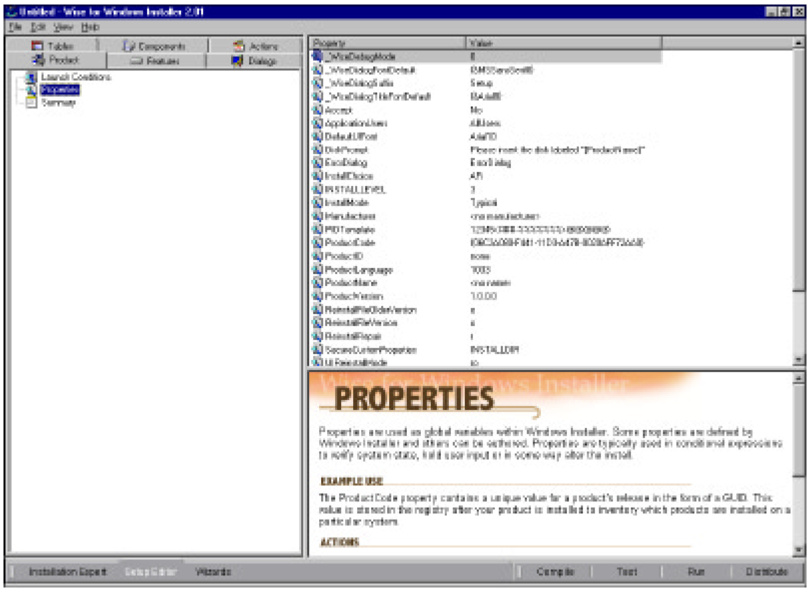
The Features and Dialogs tabs let you define your features and control which dialog boxes are displayed during installation. Using the dialog editor you can create your own custom dialog boxes, or customize any of the built-in dialog boxes. The Actions tab lets you edit existing actions or add custom actions. While it's unlikely you will need to change any existing actions, there are times when you will want to add custom actions. Custom actions let you run an EXE or call DLL functions as part of your installation. You can even run VBScript or JScript files. The Tables tab lets you edit the Windows Installer database tables directly, and is for those who are intimately familiar with the Windows Installer SDK.

Wizards

Wise for Windows Installer includes six wizards to guide you through various tasks. The wizards are described in Figure 6.

Documentation

Wise for Windows Installer comes with a 132-page *Getting Started Guide* that provides a great introduction to using the product. The description of each feature begins with an overview and continues with step-by-step instructions. If you've had no prior exposure to Windows Installer, you will find a few places in the manual where you will be scratching your head trying to understand some of the terms and concepts. To avoid this, download the Windows Installer SDK and read the introductory sections in the Help file, or visit the Windows Installer Web site and read the introductory documentation there. An extensive online Help file gives you context-sensitive help in both the Installation Expert and the Setup Editor.



Property	Value
_WiseDebugMode	0
_WiseDialogFontDefault	[MSSansSerif8]
_WiseDialogSuffix	Setup
_WiseDialogTitleFontDefault	[Arial8]
Accept	No
ApplicationUsers	AllUsers
DefaultUIFont	Arial10
DiskPrompt	Please insert the disk labeled "[ProductName]"
ExitDialog	ExitDialog
InstallChoice	AR
INSTALLLEVEL	3
InstallMode	Typical
Manufacturer	<no manufacturer>
PIDTemplate	12345<###-%%%%%%%>@@@@@
ProductCode	[illegible]
ProductID	none
ProductLanguage	1033
ProductName	<no name>
ProductVersion	1.0.00
ReinstallFileOlderVersion	o
ReinstallFileVersion	o
ReinstallRepair	r
SecureCustomProperties	INSTALLDIR
UI ReinstallMode	is

Wise for Windows Installer

PROPERTIES

Properties are used as global variables within Windows Installer. Some properties are defined by Windows Installer and others can be authored. Properties are typically used in conditional expressions to verify system state, hold user input or in some way alter the install.

EXAMPLE USE

The ProductCode property contains a unique value for a product's release in the form of a GUID. This value is stored in the registry after your product is installed to inventory which products are installed on a particular system.

ACTIONS

Figure 5: The Setup Editor.

Wizard	Description
Windows 2000 Verification	This wizard scans your installation to see if it meets Microsoft's requirements for a Windows 2000 installation. If it doesn't, error messages identify the areas of non-compliance.
Import SMS or Wise Installation	This handy wizard lets you import a Microsoft Systems Management Server installation script or a Wise Installation System installation script, and converts it to run in Windows Installer.
Run Application Watch for Loaded Files	This wizard lets you run your application while Wise for Windows and Windows Installer watches to see which files are loaded, and uses this information to build an installation.
Import VB Project	Use this wizard to import a Visual Basic 5.0 or 6.0 project file and create an installation.
Setup Capture	This wizard examines your system before and after you install an application, and builds an installation based on the changes.
Patch Wizard	This wizard allows you to build an installation that will update an existing installation on a user's computer.

Figure 6: The six wizards in Wise for Windows Installer.

Informant Fact File

Windows Installer is a giant step forward in making software installations safe, and Wise for Windows Installer is the perfect tool to let you take advantage of this new technology. The installation expert will make anyone who has used Wise 7 or 8 feel right at home, and help new users build their installations with ease. This is the future of Windows software installation.

Wise Solutions, Inc.
5880 N. Canton Center Rd., Suite 450
Canton, MI 48187

Phone: (800) 554-8565
Web Site: http://www.wisesolutions.com
Price: US$795

Conclusion

Windows Installer is a giant step forward in making software installations safe, and Wise for Windows Installer is the perfect tool to let you take advantage of this new technology. The installation expert will make anyone who has used Wise 7 or 8 feel right at home, and help new users build their installations with ease. The ability to create installation executables that will automatically install Windows Installer on any machine lets you switch to Windows Installer now for all of your installations. If you've been using Wise products, the import wizard makes the move to Windows Installer very easy. Again, Wise Solutions has done a superb job of providing both ease and power in a single product. The Setup Editor provides a fast, intuitive interface that will take you all the way to the lowest level of building an installation: editing the Windows Installer tables directly. This is the future of Windows software installation. ∆

Bill Todd is president of The Database Group, Inc., a database consulting and development firm based near Phoenix. He is co-author of four database programming books, author of over 60 articles, a Contributing Editor of *Delphi Informant Magazine*, and a member of Team Borland, providing technical support on the Borland Internet newsgroups. He is a frequent speaker at Borland Developer Conferences in the US and Europe. Bill is also a nationally known trainer and has taught 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.

Delphi 5 Developer's Guide

Since Delphi first appeared, two books have competed for the role as the top general Delphi book. One of them is Marco Cantù's *Mastering Delphi*; the other is *Delphi 5 Developer's Guide* by Steve Teixeira and Xavier Pacheco, which I recently had the pleasure of reading.

In previous reviews, I've praised earlier editions (particularly the Delphi 4 version), pointing out the expertise of Teixeira and Pacheco: Both worked for Inprise/Borland in the early Delphi days and are able to bring the insightful perspective of an "insider" to bear upon this work; at the same time, both are now active as independent Delphi developers, which gives them the practical experience to address many real-world issues.

All of the qualities I praised in the previous edition are present in *Delphi 5 Developer's Guide*, including the wealth of valuable tips. The organization of this newest edition of *Developer's Guide* is very similar to the Delphi 4 edition, and it has grown considerably in length to well over 1,500 pages — not counting the additional 500 pages on the CD-ROM. This is truly a Delphi encyclopedia!

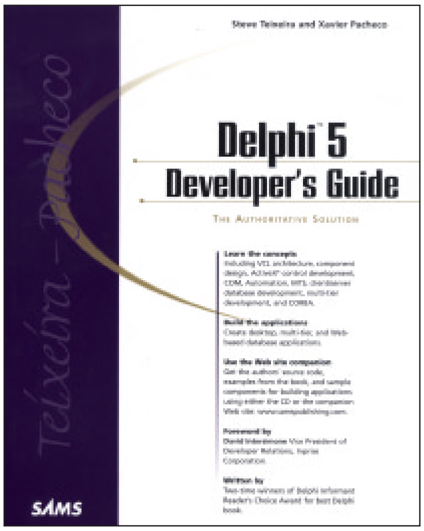
The book is divided into five parts. Part 1, "Essentials for Rapid Development," provides an introduction to Windows programming using Delphi 5. Part 2 covers a plethora of "Advanced Techniques," most of which are essential to successful development. Part 3, "Component-Based Development," introduces the Visual Component Library. Part 4 provides an introduction to database programming in Delphi. And finally, Part 5 explores database programming in more detail, getting into client/server issues and some of the new technologies available in various versions of Delphi.

In last year's edition, I especially appreciated the chapter on Object Pascal. Thankfully, the authors removed nothing, but rather made some subtle changes that made the material even more accessible, especially for developers moving to Delphi from Visual Basic, C++, or another language. Many of you will recall the numerous extensions to the Object Pascal language in Delphi 4, such as dynamic arrays and function overloading. That entire discussion remains in this volume. Chapter 4, "Application Frameworks and Design Concepts," is a favorite of mine as an excellent introduction to good programming practice.

Part 2, "Advanced Techniques," includes all the topics covered previously: graphics programming, printing, multimedia programming, working with files, and MDI applications. I still consider the chapters on working with dynamic link libraries and multithreading to be among the best I've seen. Chapter 13, "Hard-Core Techniques," and Chapter 14, "Snooping System Information," are must-reads for the intermediate-level developer who aspires to enter the ranks of advanced developers. One of the delightful topics in Chapter 13 is using the built-in assembler. The authors explain the techniques involved, and give excellent advice on when to use assembler and when not to. Six of the 12 chapters in Part 2 are presented in Adobe Acrobat format on the CD-ROM that accompanies the book.

Part 3 begins with an overview of the Visual Component Library (VCL) and contains an expanded discussion of Run-time Type Information (RTTI). It discusses many component creation topics, including writing component editors, working with packages, and using the *TCollection* class in building a component. Of course, the new Delphi 5 feature Property Categories is included. If you're interested in working with OLE, COM, or ActiveX, you should find Chapter 23, "COM-Based Technologies," to be an excellent introduction — especially with its extended treatment. This section ends with a great chapter on CORBA development.

Part 4 provides a thorough introduction to database programming, covering the essential Table, Query, and StoredProc components. It also includes a discussion of client/-server programming and Internet database issues, as well as chapters on WebBroker (written by Nick Hodges), and MIDAS development (written by Dan Miser). Even more than before, I consider this to be one of the very best introductions to Delphi database development in a general Delphi work. With Part 5, "Rapid Database Application Development," the excursion into the realm of Delphi database development continues. Here, the authors provide a solid model of building client/server and desktop applications.



My recommendation? If you don't own this book, buy it! The coverage is comprehensive, the writing style is easy to follow, and the examples have many practical applications. This is a Delphi encyclopedia that deserves a place on every developer's bookshelf.

— *Alan C. Moore Ph.D.*

Delphi 5 Developer's Guide by Steve Teixeira and Xavier Pacheco, SAMS Publishing, 201 W. 103rd St., Indianapolis, IN 46290, http://www.samspublishing.com.

ISBN: 0-672-31781-8
Price: US$59.99 (1,556 pages, CD-ROM)

Be Resourceful

If you routinely (no pun intended) use the keyword **resourcestring** in your programs, read no further. This article would merely be "preaching to the choir." But for those of you who are asking: "What in the Dickens is **resourcestring**?" read on.

In the "olden" days of Delphi programming (prior to version 4), there were basically two ways of using strings in your programs. You could either embed them into the source code using string literals:

```
MessageDlg(
  'Leave your stinkin' mitts off that button, fool!',
  mtError, [mbOK], 0);
```

Or, you could create a text file with an .RC extension, such as:

```
STRINGTABLE DISCARDABLE
{
   1 "Dialog Expert"
   2 "Dialog Expert from demonstration Expert DLL."
   3 "Application Expert"
   4 "Application Expert from demonstration Expert DLL"
   5 "&Create"
   6 "&Next"
   7 "An application name is required!"
   8 "The application name is not a valid identifier."
   9 "The path entered does not exist."
  10 MAIN.PAS"
  11 "MAIN.DFM"
  12 "MAIN.TXT"
  ...
  // Variable names.
  20 "StatusLine"
  ...
}
```

Then all you had to do was compile it into a resource file, add it to a Delphi project or unit, compile it using the command-line tool Brcc32.exe, and then programmatically extract the strings in the appropriate places in your code using the *LoadStr* function. That may have seemed a bit too much of a bother, so you may have stuck with the tried and true — and now tired — old way of introducing string literals ad infinitum into your source code.

The **resourcestring** keyword has come to the rescue. It gives us the best of both worlds: the simplicity (almost) of string literals, and the convenience of storing all strings in a central location. Also, using **resourcestring** provides better memory management, because the strings in the **resourcestring** section are saved as resources associated with your application.

To take the plunge into the brave new world of using the **resourcestring** keyword, simply add a unit to your project, name it ResStrngs (or whatever), and then add any string literals (especially those the user would see — contents of string lists, feedback, error messages, etc.) in the **interface** section of the unit, like this:

```
unit ResStrngs;

interface

resourcestring
  // Famous military personalities.
  SGeneralElectric =        'General Electric';
  SGeneralMills =           'General Mills';
  SGeneralUsage =           'General Usage';
  SGeneralHospital =        'General Hospital';
  SGeneralLedger =          'General Ledger';
  SGeneralProtectionFault = 'General Protection Fault';
  SGeneralSQLError =        'General SQL Error';
  SGeneralLeeSpeaking =     'General Lee Speaking';
  SCorporalPunishment =     'Corporal Punishment';
  SSgtFury =                'Sgt. Fury';
  SSgtCarter =              'Sgt. Carter';
  SSgtSchultz =             'Sgt. Schultz';
  SSargentShriver =         'Sargent Shriver';
  SCaptKangaroo =           'Capt. Kangaroo';
  SCaptUnderpants =         'Capt. Underpants';
  SColonelKlink =           'Colonel Klink';
  SPrivateBenjamin =        'Private Benjamin';
  SPrivateProperty =        'Private Property';
  SLeftenantDan =           'Leftenant Dan';
  SMutineerChristian =      'Mutineer Christian';
  SAtlantaHawks =           'Atlanta Hawks';
  // Kindly reminders.
  SDontSleepInTheSubwayDarlin =
    'Don''t sleep in the subway darlin''';
  // Additional silly strings left as a reader exercise.

implementation

end.
```

Add this unit to the **implementation uses** clause of any unit that refers to any of its strings. Then access them in this way:

```
if ItIsPetulasVirtualHusband and HeIsLate then
  MessageDlg(SDontSleepInTheSubwayDarlin,
             mtInformation, [mbOK], 0);
```

For an example of how Borland/Inprise/Corel (will they be giving away Bic lighters at the next conference?) uses resource strings, see consts.pas, dbconsts.pas, etc. in ..\source\vcl.

It's also beneficial to place your strings in a **resourcestring** section because programmers are often not the best people to write feedback and error messages that will be seen by users. They tend to be either too technical, "An unexpected error occurred in module xyz while attempting to spawn a thread;" not informative enough, "Post the changes before proceeding;" and/or a little testy, "Error! You can't [whatever] until you [whatever]."

Separating the message strings to a discrete file makes it easier to hand them over to someone with the skills necessary to write user messages (consulting with the programmers as to what exactly each message is meant to convey, of course). If you don't want non-programmers mucking about in your .pas file directly, you can copy and paste the existing resource strings to a text file and then, after they've finished polishing your prose, copy their changes over the resource string constants.

Last, but not least, it is far easier to internationalize (genericize) and then localize (specificize) your applications when the strings the user will see are collected in one place. With Delphi's ITE (Integrated Translation Environment), the process of internationalizing and then localizing your application's strings is semi-automated. Using the ITE, you create a separate resource .dll for each target language. If you deploy multiple .dlls, the correct one is automatically loaded according to the locale of the computer on which your program runs.

The primary tools in the ITE are the Resource DLL Wizard (File | New | Resource DLL Wizard) and the Translation Manager, where translations can be entered. See "Integrated Translation Environment" in Delphi Help for the specifics.

Besides the ITE included with Delphi, there are third-party offerings relative to internationalization and subsequent localization of Delphi applications. I prefer to use something that comes "in the box." As long as it works well, of course, as the ITE seems to. ∆

— *Clay Shannon*

Clay Shannon is a Delphi developer for eMake, Inc. in Post Falls, ID. Having visited 49 states (all but Hawaii) and lived in seven, he and his family have finally settled in northern Idaho, near beautiful Lake Coeur d'Alene. The only spuds he has seen in Idaho have been in the grocery, and most of those are from Oregon and Washington. Clay has been working (almost) exclusively with Delphi since the release of version 1, and is the author of Developer's Guide to Delphi Troubleshooting *(Wordware, 1999). You can reach him at BClayShannon@aol.com.*

An Interview with Robert A. DelRossi

Robert A. DelRossi is president of TurboPower Software Co., one of the oldest and most successful third-party producers of Delphi tools. In the most recent *Delphi Informant Magazine* Readers Choice Awards (published in the April, 2000 issue), TurboPower scored big — even bigger than last year. In 1999, TurboPower won four awards: Async Professional won the Best Connectivity Tool, Memory Sleuth won Best Utility, Orpheus 3 won Best VCL, and SysTools won Best Add-in Library. This year, TurboPower came in first place in five categories and placed second in two others. Memory Sleuth's new incarnation, Sleuth QA Suite, won the award for Best Testing/Debugging Tool, with each of the other three 1999 winners repeating. Abbrevia won the award for Best VCL Component, and two other products — FlashFiler and LockBox — came in second in their respective categories. Especially remarkable was *Delphi Informant Magazine*'s decision to give a new award this year, Company of the Year, won by — you guessed it — TurboPower. You can find more information about TurboPower products at its Web site, http://www.turbopower.com.

DelRossi has not always been the president of this company. He began many years ago as a TurboPower customer. After he joined the company as its first marketing director, he became vice president, and, finally, its third president.

DI: You've had a varied and impressive career at TurboPower. Could you elaborate a bit more about your experiences at TurboPower? What advice do you have for the developer who aspires to embark upon a similar career path?

DelRossi: For starters, TurboPower is a great place to work. That's always been the case, and our management team has dedicated itself to keeping it that way. Many companies lose the positive corporate culture they start with, especially as they grow. At TurboPower we regularly review not just how well the products are doing, but also how the staff is doing, and we make adjustments to meet the challenges of a growing company. Good communication among team members — the engineers and our terrific operations group — is a key to our success.

Robert A. DelRossi

For me personally, and this is especially true having come from a different kind of business, I've learned that managing highly intelligent people requires a certain amount of flexibility. Software development is still not 100 percent science. There's a large dose of art involved too. Managing risk, practicing sound business fundamentals, and applying proven project management strategies is a big part of successful software company management, I believe.

DI: What do you feel was your biggest challenge as a developer; as a manager?

DelRossi: As a developer, keeping up with the incredible pace of change in our industry is the biggest challenge. For most of us, the older you get, the harder it seems to keep up with the amazing degree of change. That's why TurboPower is successful, I think. We make it easier to get the latest technology advances into your programs without having to learn everything about them first.

Without question, the biggest challenge any software executive faces today is finding and retaining qualified engineers. When I first became president I imagined that the biggest challenges would be financial, but it's really finding the right people to maintain your company's growth, and then keeping them engaged.

DI: You must be gratified with the great success that TurboPower has enjoyed in recent years, particularly the unprecedented showing in this year's *Delphi Informant Magazine* Readers Choice Awards. To what do you attribute this success? Is there one factor — the quality of the software, the documentation, the support — that stands above all the others as the key to their popularity with users?

DelRossi: You probably won't be surprised to hear me say that I believe our success comes from several factors. I think our customers see us as providing more than just good code. When they buy one of our products, they're really making us a partner in their projects. If *they* succeed, *we* succeed — plain and simple. And if they fail, they'll look to us and ask why — particularly if they feel that our products let them down. So, we try to produce a unified set of services: great code, great documentation, great support, and a willingness to learn, accept criticism, and help our customers achieve their very best work.

DI: I'd like to explore one of your newer products, one that I am currently in the process of reviewing: Sleuth QA Suite. Many in the Delphi community have been encouraging you for some time to fill the gap that used to exist and create a solid Delphi profiler. What were some of the challenges in developing this tool that might have contributed to it taking longer than you expected?

DelRossi: It seemed like everyone was asking us to develop a professional profiling tool for Delphi and C++Builder! For years, the big toolmakers have focused solely on Microsoft developers. The key tools, like NuMega BoundsChecker, really ignored the fact that real-world development was being done with Borland compilers. Over time, BoundsChecker, and some similar products were ported to work with Borland compilers, but in our opinion, they never quite made the grade.

Like a lot of our products, Sleuth QA Suite was born out of a very real need we had internally. We needed a bug detection tool and profiler that was an expert when it came to the VCL. The trouble is, getting all that VCL knowledge into Sleuth QA Suite took a lot longer than we expected. Plus, we kept coming up with additional capabilities that we felt were imperative to Sleuth QA Suite's success.

In the end, I think we came up with a great product, and to tell the truth, though it was a lot later than we would have liked, rushing it out the door wouldn't have been the answer. Building quality products sometimes takes a little longer than any of us expect.

DI: I'd like to change the topic a bit and talk about some of the other folks who work at TurboPower. For example, I've known Julian Bucknall for several years and have been very impressed with his articles and book chapters. I understand he is currently in the process of finishing a book on algorithms. Of course, he is just one of several of your engineers active as Delphi writers. Talk a bit about some of these active writers and some of the reasons you not only allow this but encourage such "extra-curricular" activities.

DelRossi: I've been at TurboPower four years now and I'm still awed by our programmers. I was a programmer too, once upon a time, but I was never anywhere near the caliber of our engineering staff.

One reason I think we've been so fortunate at attracting such high-end developers is that we give our developers a chance to participate in every level of the product development cycle, from design to marketing. Not everyone likes every part of the process, of course, but there's much to be learned and at the end of the day, a greater appreciation for how each phase of the cycle develops.

Another area, as you say, is to encourage our engineering staff to write and give lectures. Again, not all of our engineers want to do these things, but for those that enjoy it, this allows for a sharing and exchange of knowledge; it's great for them, as well as for the whole company.

DI: By all reports, the latest version of Delphi seems to be one of the most solid ones to appear in several years. Does this seem to be an accurate assessment, and if so, do you feel it might help Delphi's market share begin to expand?

DelRossi: We're very satisfied with this version of Delphi, although there's lot more that could be done. Of course, all those gaps give us plenty to do!

DI: Some component-producing companies also produce versions for other tools like Visual Basic. Have you ever considered this? What do you see as the advantages and disadvantages?

DelRossi: We've certainly considered expanding our product line to support Visual Basic, and COM developers generally. Many of our customers have for one reason or another — perhaps at the request of a key client — moved away from Delphi to embrace COM, and they tell us they'd like to use our products too.

Expanding our products to support all kinds of developers is an important part of my vision for TurboPower. As a matter of fact, we've already begun including COM implementations in some of our products, including Abbrevia and SysTools 3. In my view, COM support is important for a number of reasons. Naturally, it exposes TurboPower products to a wider audience of Visual Basic and Visual C++ developers, but it also makes some of our technology available for Web development. In fact, the COM object in SysTools 3 is used extensively on our own Web site.

DI: In previous interviews with developers I have always included a question on operating systems. Windows is very popular right now as a computing environment, but Linux is becoming such a strong competitor that Inprise/Borland has decided to develop tools for this environment. I noticed that you included an article on this topic in the February, 2000 issue of your newsletter, *Powerlines*. Please share some of your thoughts on the future growth of Linux and how you see this impacting TurboPower.

DelRossi: We're incredibly excited about the potential for Linux, particularly with the advent of Borland's Kylix project. For some time now, I've regarded the Linux marketplace as the Windows market was back before Visual Basic. In those days, creating Windows programs was essentially a labor of love. You had to learn all those API calls and bring out the old command-line C compiler. Then Visual Basic arrived and millions of developers turned to it as an easy way to build Windows applications. Love it or hate it, Visual Basic changed everything for Windows. Even programmers who wouldn't have considered programming in Basic again started using Visual Basic, because it was so much easier to build Windows programs with it.

The Linux development world is also waiting for its Visual Basic and, quite frankly, we think Kylix may be it. As it was back then, there are probably many C and C++ programmers that think their Pascal days are behind them. But Kylix may represent such a fundamental improvement in Linux-targeted programming that Pascal's glory days could be coming back. In a big way!

DI: I'd like to conclude by talking a bit about Inprise. We've witnessed a lot of changes in recent years — certainly some ups and downs. Among other remarkable developments, there is now a significant merger with Corel Corporation on the horizon. What is your perspective on some of the new developments at Inprise? How will this affect the future of Delphi and your company?

DelRossi: Inprise has always had great development talent and terrific developer products. Of course, my crystal ball is no better than anyone else's. But from where I stand, Inprise has a better outlook now than at any time in its recent history. Naturally, much of that will depend on the rate of Linux adoption.

DI: Thank you very much, and best of luck with all your future endeavors. ∆

— 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