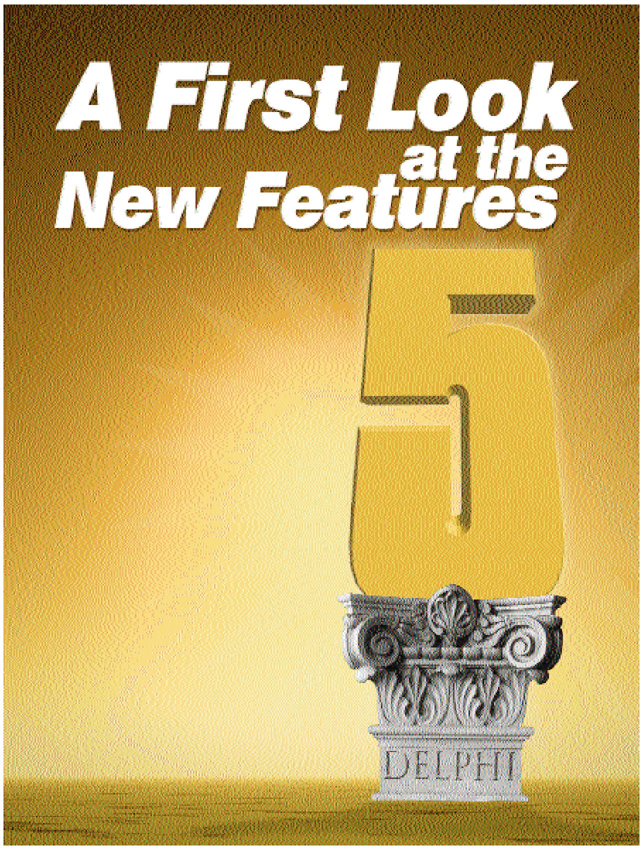
August 1999, Volume 5, Number 8



Cover Art By: *Darryl Dennis*

Delphi TOOLS

New Products and Solutions

PractiCalSoft Releases Components for Delphi

PracticalSoft announced the release of three shareware components for Delphi: *TArrayBitmap*, *PanoMax Panorama*, and the *Organic Shape Pack*.

The *TArrayBitmap* class is a descendant of *TBitmap* that provides high-speed access to 24-bit bitmap pixels, making it ideal for multimedia applications with graphic effects.

The *PanoMax Panorama* is a component for Delphi 3 and 4 that renders real-time views of panoramic (360-degree) images, giving a virtual reality-like immersion feeling.

The Organic Shape Pack contains components that have the ability to shape themselves according to a bitmap used as a mask, making them ideal for building interfaces. The Organic Shape Pack supports automatic skin saving and retrieval.

PracticalSoft
Price: *TArrayBitmap*, US$10; *PanoMax Panorama*, US$59 (compiled Delphi unit) or US$160 (full Delphi source); Organic Shape Pack, US$24 (Organic Shape Form), US$28 (Organic Shape Button), and US$20 (Organic Shape Image).
E-Mail: practicalsoft@usa.net
Web Site: http://practicalsoft.hypermart.net

Blaise Announces Blaise Compound Components

Blaise Software Inc. announced *Blaise Compound Components*, a set of over 50 standard, additional, and data-aware components that have other components attached.

A Compound Component is an object made up of multiple components, allowing one to encapsulate properties, methods, and events from many components into one. An example of this is a *TEdit* combined with a *TLabel* for a caption. The user of this new object would be able to access attributes of both individual components through one compound component.

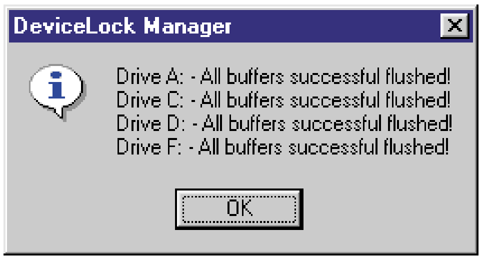
All Compound Components are created from the same base class, over which the developer has control. This allows one to add the same functionality to all components at the same time.

Ease-of-use features include component editors and naming components.

Blaise Software Inc.
Price: US$200
E-Mail: sales@blaisesoftware.com
Web Site: http://www.blaisesoftware.com

SmartLine Releases DeviceLock

SmartLine, Inc. announced the release of ***DeviceLock***, an NT service for restricting access to local devices running Windows NT. DeviceLock eliminates the need for physical locks and has a number of advantages. It is easy to install and administrators can have instant access from the remote computers when necessary.


DeviceLock Manager

Drive A: - All buffers successful flushed!
Drive C: - All buffers successful flushed!
Drive D: - All buffers successful flushed!
Drive F: - All buffers successful flushed!

In addition, DeviceLock allows the administrator of the machine or domain to designate user access to floppy drives, other removable media, CD-ROM drives, or serial and parallel ports.

DeviceLock requires Windows NT 4.0 (for x86 or ALPHA platforms), 8MB of RAM, and a hard drive with 1MB of hard disk space.

SmartLine, Inc.
Price: US$40 for a single-user license; site and world licenses are available.
Phone: (+7 095) 366-2193
Web Site: http://www.protect-me.com

Pervasive Delivers Crystal Reports 7 for Pervasive

Pervasive Software Inc. announced *Crystal Reports 7 for Pervasive*, a Windows-based query and reporting tool for software applications based on Pervasive.SQL, providing interactive Web reporting, report integration, and report generation capabilities from virtually any data source. The application is based on Seagate Software's Seagate Crystal Reports 7 desktop and Web reporting tool. Crystal Reports 7 for Pervasive enables users of applications built on Pervasive.SQL (including Pervasive's Btrieve database code) to create business tools such as reports, lists, letters, forms, and labels. The release includes Java-based Web query and analysis, support for server-side processing, and the ability to import legacy reports.

A copy of the Pervasive.SQL Workstation engine ships with Crystal Reports 7 for Pervasive and includes Pervasive's DDF Ease, which makes it easier to build, maintain, and edit the required DDF files.

The Crystal Reports engine is a true DLL that can be integrated into Windows- or Web-based applications to manage a broad range of controls and class 1 libraries, including ActiveX (OCX) controls (16- and 32-bit), Visual Basic Custom controls (VBX), Microsoft Foundation Class Library with AppWizard for Visual C++, Delphi VCL, and Report Designer for Visual Basic 5.0 and 6.0.

Crystal Reports 7 for Pervasive offers new and enhanced features for improved programming control and comprehensive functionality, including the Document Import Tool, Field Mapping, On-Demand Subreports, Geographic Mapping, Running Totals Expert, and the JavaBean Viewer.

Pervasive Software Inc.
Price: US$345 for single-user license.
Phone: (800) 287-4383 or (512) 231-6000
Web Site: http://www.pervasive.com

IDEAL Ships Virtual Print Engine 3.0

IDEAL Software announced it is shipping *Virtual Print Engine 3.0*, its solution for developers needing dynamic, precise control over printed output from their applications.

Virtual Print Engine provides developers using Delphi, C/C++, Visual Basic, Visual FoxPro, and other languages the ability to create preview and print reports, rich documents, drawings, etc. by calling functions during run time.

Through code, objects such as text, lines, polygons, images, and 21 barcodes can be positioned, rotated, and scaled with 0.1 mm precision on any number of pages. The vector graphics offer a free-scalable WYSIWYG preview and printer-independent output.

New features in version 3.0 include RTF parsing, integrated chart objects, interactive objects, user-defined objects with direct access to the Windows Device Context, export of pages to all common image types, scale-to-gray technology, e-mail and fax, and more. *Virtual Print Engine 3.0* is available for US$429. For more information call +49 (0)2131 9800-23, or visit IDEAL's Web site at http://www.idealsoftware.com.

Discmatic Offers Multi-drive Duplicators

Discmatic, a division of CBC (AMERICA) Corp., introduced two new multi-drive CD duplicators: the ***MDX7000*** and ***MDX3000***. The MDX7000 can produce up to 21 full CDs per hour, and the MDX3000 can produce up to nine per hour.

The MDX7000 and MDX3000 tower CD duplicators combine Discmatic's EZ-ONE controller engine with seven and three CD drives, respectively. A new SCSI bus allows a faster data transfer rate and permits copying of up to seven discs simultaneously. In addition, both units can copy directly from CD to CD rather than having to work through the hard drive.

The core logic for both units is stored on an advanced flash ROM. In addition, the internal firmware can be upgraded via a firmware CD or by downloading the firmware file from the Discmatic Web site.

Features include Image Management, which allows files to be loaded, deleted, renamed, and undeleted with ease, and Hard Disk Management for efficient use of hard disk capacity.

An Audio Compilation feature allows selected tracks from a variety of discs to be assembled and stored on the internal hard drive as a CD image. From there, a new master is created. For larger duplication jobs, Discmatic will offer the option of connecting up to eight MDX7000 and MDX3000 duplicators via a SCSI channel to build a networked duplication system. Eight MDX7000s connected in this way would be capable of producing up to 56 discs at once.

Discmatic
Price: MDX3000, from US$3,035 with one read drive and two write drives; MDX7000, from US$4,835 with one read drive and six write drives.
Phone: (800) 422-6707
Web Site: http://www.discmatic.com

Epsilon Squared Releases InstallWatch 1.1

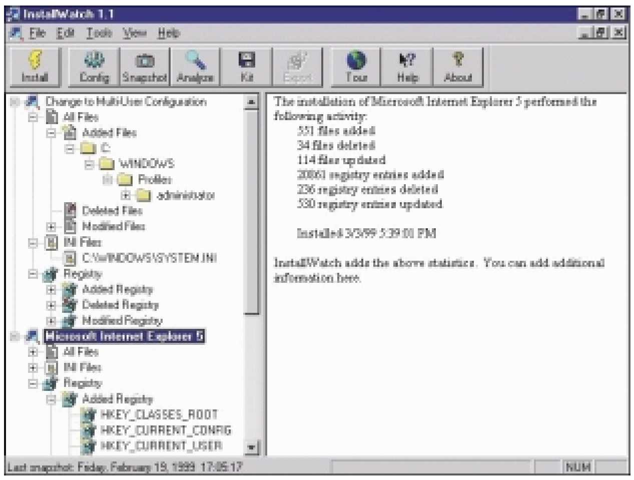
Epsilon Squared, Inc. released ***InstallWatch 1.1***, a Windows 95/98/2000 and Windows NT 4.0 application that records all changes made to a PC during installation of software from the Web, CD-ROMs, or floppies. InstallWatch 1.1 can also record all modifications made during configuration changes and hardware additions.

InstallWatch 1.1 will deliver detailed information about the specific actions taken during software installations. This information is stored permanently and can be reviewed with an Explorer-like interface, printed, or exported to HTML format for posting to an intranet.

InstallWatch 1.1 records all additions, deletions, and modifications to the Windows registry, files, and INI files. InstallWatch 1.1 will monitor a system silently until it detects the installation of a software application. InstallWatch's InstallWizard will immediately begin and guide the user through the process of capturing the details of the installation.

InstallWatch 1.1 can also be used in a manual mode to detect any changes to a PC. Users about to install hardware that will update drivers or settings can use InstallWatch 1.1 before and after the hardware install to save a record of the changes in the InstallWatch database. Network administrators can use the manual mode to detect changes to the server every day, logging activities to the InstallWatch database.

Epsilon Squared, Inc.
Price: US$79 per license.
Phone: (941) 752-1470
Web Site: http://www.installwatch.com



```
The installation of Microsoft Internet Explorer 5 performed the
following activity:
        551 files added
        34 files deleted
        114 files updated
        20061 registry entries added
        236 registry entries deleted
        530 registry entries updated

        Installed 3/3/99 5:39:00 PM

InstallWatch adds the above statistics.  You can add additional
information here.
```

Last snapshot: Friday, February 19, 1999 17:05:17

Xceed Announces Xceed Zip Compression Library Version 4.0

Xceed Software Inc. announced the release of version 4.0 of the *Xceed Zip Compression Library*. The heart of the new library consists of a new, multi-threaded ZIP compression engine.

Version 4.0 offers features such as reading/writing multi-part ZIP files directly to the hard drive, streaming ZIP and unzip compression, custom include/exclude filters, Unicode API calls when running on NT, reading/writing Unicode filenames and NT file attributes and security permissions, automatic ZIP file error recovery, file previewing, which combines the entire library into one fully self-contained ActiveX control.

Xceed Zip v4.0's new multi-threading ZIP compression engine offers faster overall zipping performance. It supports both single-threaded (STA) and multi-threaded apartment (MTA) models, and has low memory requirements.

Xceed Software Inc.
Price: US$299.95; US$399.95 with the Xceed Zip Self-Extractor Module version 1.2.
Phone: (800) 865-2626 or (450) 442-2626
Web Site: http://www.xceedsoft.com

DBI Technologies Announces Solutions::Explorer 1.0

DBI Technologies Inc. announced the release of ***Solutions::Explorer 1.0***, the latest product in the Solutions::Series. At the heart of the package is the *ctExplorer* ActiveX control, which makes creating multiple view forms easier. This meta-component combines a tree view, list view, html view, list bar, tabs, and splitter objects into a single component.

The Solutions::Explorer package includes other components, such as *ctButton*, an enhanced transparent button control for creating Explorer-type toolbars; *ctFile*, a common Open/Save File dialog box control; *ctFrame*, a frame control for creating toolbars; and *ctHypLnk*, a hyperlink label control for embedding URLs and other links.

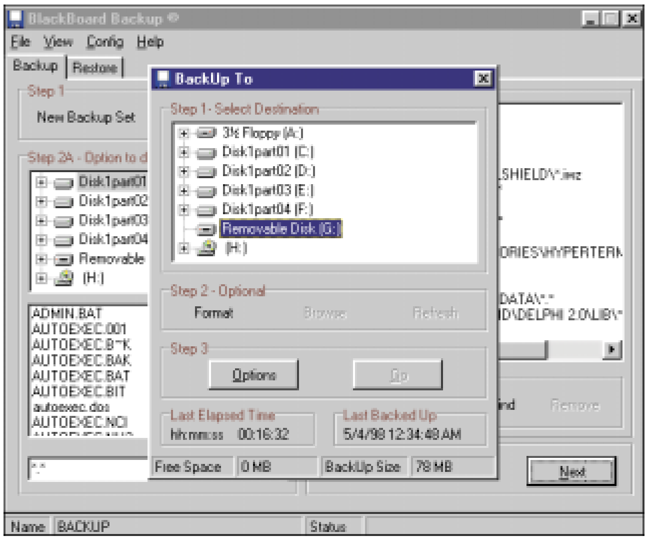
Solutions::Explorer is compliant with multiple visual platforms, and recreates the Microsoft standard look and feel for any application.

DBI Technologies Inc.
Price: US$399 for a single developer (introductory offer; prices may change).
Phone: (800) 670-8045
Web Site: http://www.dbi-tech.com

Albert's Ambry Presents BlackBoard Backup 5.9

Albert's Ambry announced they are offering ***BlackBoard Backup 5.9***, a 32-bit file backup and archive with high compression and disk spanning. BlackBoard is designed to replace the backup utilities that ship with Windows 95/98/NT.

BlackBoard Backup 5.9 allows you to take advantage of high LZH5 compression, password protection, and disk spanning.



You can also back up files that have changed (incremental), as well as test file integrity.

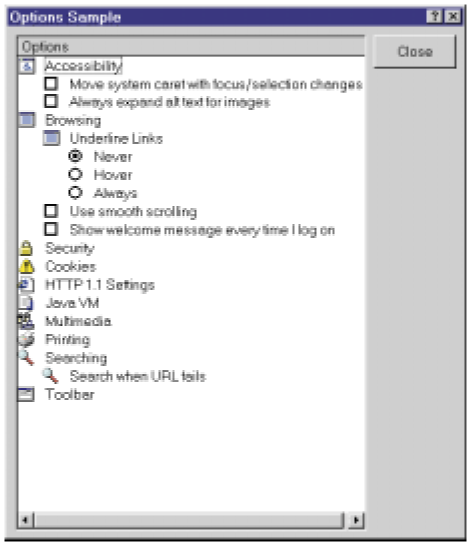
Simply drag and drop the files and folders to be backed up. Select as the backup destination any drive letter including floppy, removable, and networked drives. The program offers a display of backup size and the available space on the destination drive.

The latest release of BlackBoard Backup also includes IntelliBak to help find files that need to be backed up, and AutoBak, which can be used with a task-scheduling program to back up files automatically.

Albert's Ambry/BlackBoard Software
Price: US$29
E-Mail: dalin@blackboardsoftware.com
Web Site: http://www.blackboardsoftware.com or http://www.alberts.com/authorpages/00000492/Prod_471.htm

Softel Releases SftTree/VCL 4.0

Softel vdm, Inc. released *SftTree/VCL 4.0*, a new version of its tree control for Windows specifically designed for use with Delphi and C++Builder.



The new version includes support for virtual lists (up to two billion items), a vertical splitter bar between columns, Right-To-Left Reading support for international Windows versions, full multi-column support with cell colors, cell fonts and cell bitmaps, ScrollTips displayed during vertical scrolling, column header ToolTips, a background bitmap, item searching based on typed input, and more.

SftTree/VCL 4.0 offers hierarchical data displays, with such features as multiple columns, multiple selection, multiple text lines per item, cell wordwrap, built-in row and column headers, user-resizable columns, column re-ordering, grid lines, 3D item display, etc.

Softel vdm, Inc.
Price: US$199 for a single-developer license (includes manual, technical support, and downloadable maintenance updates).
Phone: (941) 505-8600
Web Site: http://www.softelvdm.com

table of CONTENTS

By Cary Jensen, Ph.D.

Delphi 5

A Quick Look at What's New

It's been 13 months since Delphi 4 shipped, which means it's time for another edition of the best development environment for Windows. If this is what you've been thinking, then you were not surprised when borland.com announced the latest and greatest version of its flagship programming tool at their 10th annual conference in Philadelphia this July. Even if this thought had never crossed your mind, you'll be happy with the enhancements of this release. From the IDE (integrated development environment) to Web development, from COM to database support, from team workflow to multi-language development, Delphi 5 has something new for every Delphi developer.

In this article, we'll take a tour of the new features of Delphi 5. As you might imagine, however, this article is based on a beta version. Consequently, the features that ship with the official release may be different. Also unavailable at this time is information concerning which versions of Delphi 5 will have particular features described here. For example, you can probably safely assume that Delphi 5 Enterprise (the new name for Delphi Client/Server) will have most, if not all, of these features. The Professional edition will certainly not have everything described here.

There's good news on another front as well — that of stability. Delphi 4, for all of its fabulous enhancements, was a disappointment when it came to stability. Even after three update packs (the second one measuring more than 27MB for the Client/Server edition), Delphi 4 continues to be an embarrassment. Delphi 5 is a welcome relief. It's far more stable, even in the beta version used for this article, than Delphi 4. In short, Delphi 5 should go a long way to repairing the good name of Delphi.

Enhanced IDE

The obvious updates to a development tool are those that appear in the IDE. As you can see in Figure 1, Delphi's toolbars have again received a facelift, albeit a much smaller one than we saw between Delphi 3 and Delphi 4.

Desktop settings. Delphi 5 permits you to create a number of different desktop layouts and save them for easy retrieval. A desktop layout consists of the various windows and non-modal dialog boxes that are open within Delphi. For example, by default, the Code Explorer is docked in the Editor. You might prefer that it appear undocked by default. No problem. Undock the Code Explorer, move it to where you want it to appear, then click the **Save current desktop** button on the Desktops toolbar. Delphi will ask you to name the desktop setting, and then add it to the Desktop speed settings combo-box. Anytime you want to display this custom desktop layout, simply select the setting you saved.

Most developers will have several settings they prefer, depending on the task at hand. Delphi 5 permits you to designate one of these settings as your default debugging desktop. This desktop configuration will automatically be loaded when you use the integrated debugger. Your previous desktop setting is restored once you exit the debugger.

Updated Object Inspector. There have been few visible changes to the Object Inspector over the years — until now. Delphi 5's Object Inspector is radically different in two ways. The most obvious is the inclusion of images in some property fields. For example, the field associated with a color

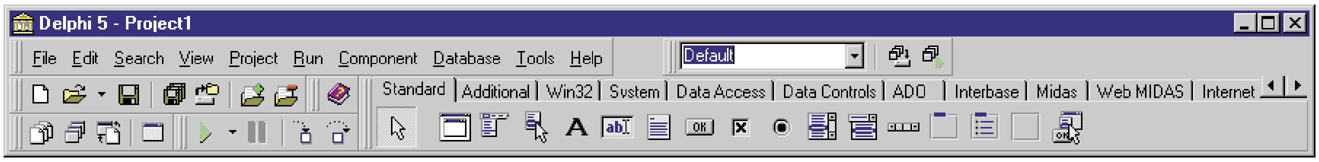
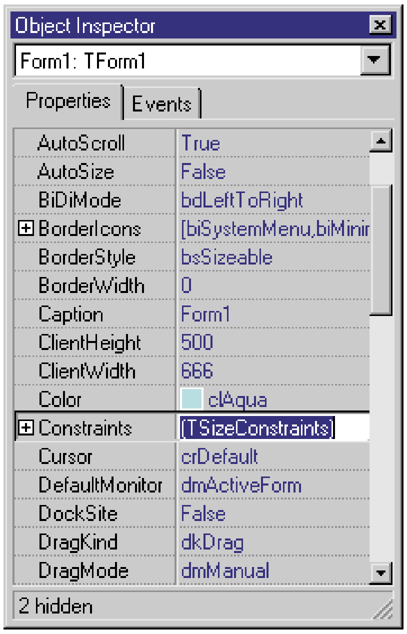

Delphi 5 - Project1
File Edit Search View Project Run Component Database Tools Help
Default
Standard Additional Win32 System Data Access Data Controls ADO Interbase Midas Web MIDAS Internet

Figure 1: Delphi 5 toolbars.



Object Inspector

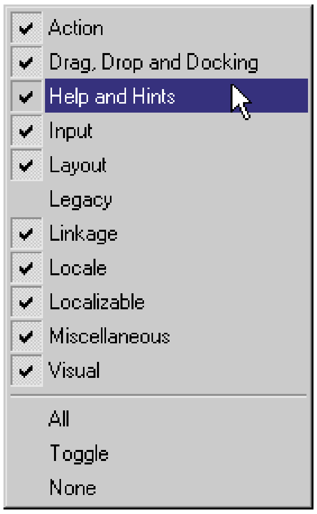
Form1: TForm1

Properties | Events

AutoScroll	True
AutoSize	False
BiDiMode	bdLeftToRight
⊞ BorderIcons	[biSystemMenu,biMinir
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	500
ClientWidth	666
Color	clAqua
⊞ Constraints	[TSizeConstraints]
Cursor	crDefault
DefaultMonitor	dmActiveForm
DockSite	False
DragKind	dkDrag
DragMode	dmManual

2 hidden

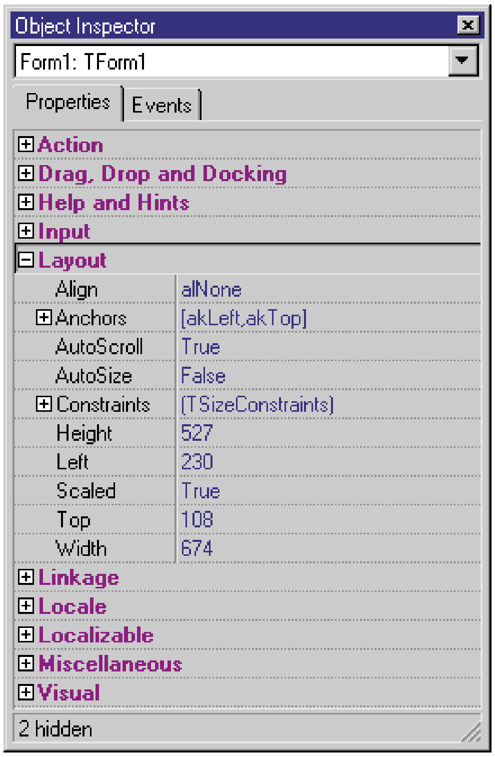
Figure 2: The Object Inspector can display custom images, such as a selected color.



```
✓ Action
✓ Drag, Drop and Docking
✓ Help and Hints
✓ Input
✓ Layout
  Legacy
✓ Linkage
✓ Locale
✓ Localizable
✓ Miscellaneous
✓ Visual

  All
  Toggle
  None
```

Figure 3: The **View** menu of the Object Inspector's right-click menu.



Object Inspector

Form1: TForm1

Properties | Events

⊞ Action	
⊞ Drag, Drop and Docking	
⊞ Help and Hints	
⊞ Input	
⊟ Layout	
Align	alNone
⊞ Anchors	[akLeft,akTop]
AutoScroll	True
AutoSize	False
⊞ Constraints	[TSizeConstraints]
Height	527
Left	230
Scaled	True
Top	108
Width	674
⊞ Linkage	
⊞ Locale	
⊞ Localizable	
⊞ Miscellaneous	
⊞ Visual	

2 hidden

Figure 4: The Object Inspector can display properties by category.

property now not only shows the name of the selected color, it displays the color as well (see Figure 2). This reflects updates to Delphi's tools API, where developers creating property editors can customize what is drawn in the Object Inspector. (See Robert Vivrette's article, "Delphi 5 Drill-Down," on page 10 for more information on this feature.)

Another new feature of the Object Inspector is revealed if you closely inspect Figure 2. At the bottom of this Inspector is the message: `2 hidden`. This message is related to views and arrangements, which provide you with alternative ways to work with the properties in the Object Inspector. To control which properties are visible in the Object Inspector, right-click it and select **View**. The **View** menu, shown in Figure 3, includes a list of checkable menu items that you can use to view or suppress individual property categories in the Inspector. This can be particularly handy when you are focusing your design efforts on a particular type of property, and don't want to be distracted by others.

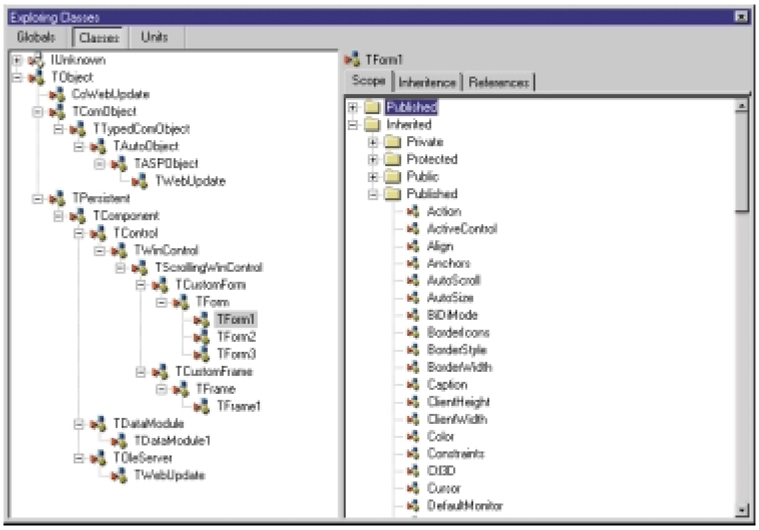
Normally, the properties you're viewing are displayed in alphabetical order in the Object Inspector. If you right-click the Object Inspector and select **Arrange | by Category**, properties will appear under expandable nodes, as shown in Figure 4. The view you select is saved with your desktop settings.

Updated dialog boxes. A number of the dialog boxes have been improved and enhanced. For example, the Project Manager dialog box now supports drag-and-drop operations. For instance, you can drag a file from the Windows Explorer and drop it into the Project Manager to add it to a project. Also, files can be dragged from one project to another.

Another major update is to the Exploring Classes dialog box (see Figure 5). This dialog box, which has seen very few changes since Delphi 1, has been radically re-designed. Although I miss some of the features that were removed, such as the ability to explore all units involved in the last compile, the new browser provides you with a structured means of exploring your project's containers (Forms, Data Modules, and Frame).

Data Module Designer. One new feature likely to be high on the "favorites" list of a good many Delphi developers is the Data Module Designer. Data modules are no longer simply non-visual forms. Instead, the data module's role as a repository for data access components has been greatly enhanced by a new interface that provides you with visual feedback concerning the roles and configurations of the contained components.

Figure 6 displays a simple data module containing several data access components. The Components page, on the right side of this window, shows the components as they previously appeared in a data module



Exploring Classes

Globals | Classes | Units

```
⊞ IUnknown
⊟ TObject
    CoWebUpdate
  ⊟ TComObject
    ⊟ TTypedComObject
      ⊟ TAutoObject
        ⊟ TASPObject
            TWebUpdate
  ⊟ TPersistent
    ⊟ TComponent
      ⊟ TControl
        ⊟ TWinControl
          ⊟ TScrollingWinControl
            ⊟ TCustomForm
              ⊟ TForm
                  TForm1
                  TForm2
                  TForm3
            ⊟ TCustomFrame
              ⊟ TFrame
                  TFrame1
      ⊟ TDataModule
          TDataModule1
      ⊟ TOleServer
          TWebUpdate
```

TForm1

Scope | Inheritence | References

```
⊞ Published
⊟ Inherited
  ⊞ Private
  ⊞ Protected
  ⊞ Public
  ⊟ Published
      Action
      ActiveControl
      Align
      Anchors
      AutoScroll
      AutoSize
      BiDiMode
      BorderIcons
      BorderStyle
      BorderWidth
      Caption
      ClientHeight
      ClientWidth
      Color
      Constraints
      Ctl3D
      Cursor
      DefaultMonitor
```

Figure 5: The Exploring Classes dialog box now sports a completely new look.

in earlier versions of Delphi. On the left is the tree view. Here you can see at a glance the major settings defined for your components, including the alias (DatabaseName) and table names assigned to the tables, and the DatabaseName associated with the database.

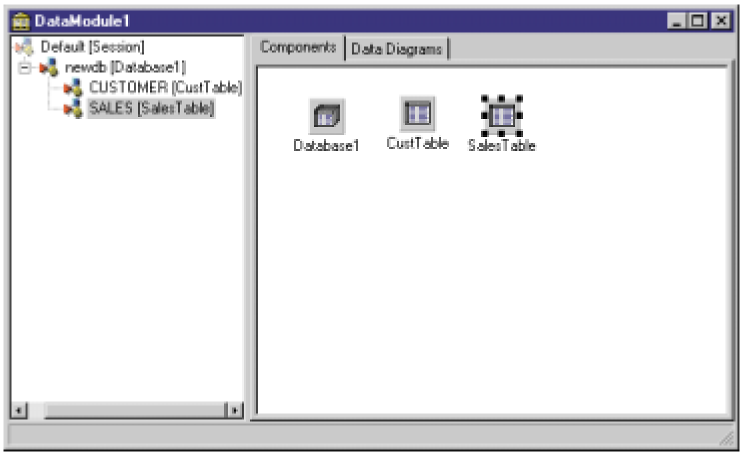


Figure 6: Delphi 5's new Data Module Designer.

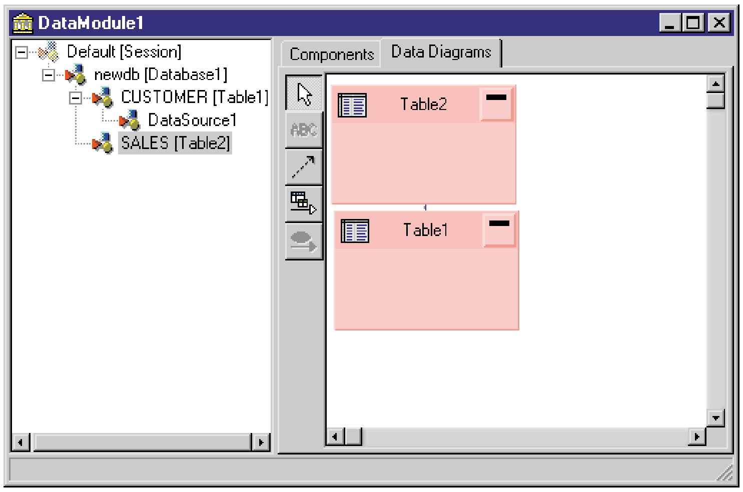


Figure 7: The Data Diagrams page of the Data Module Designer.

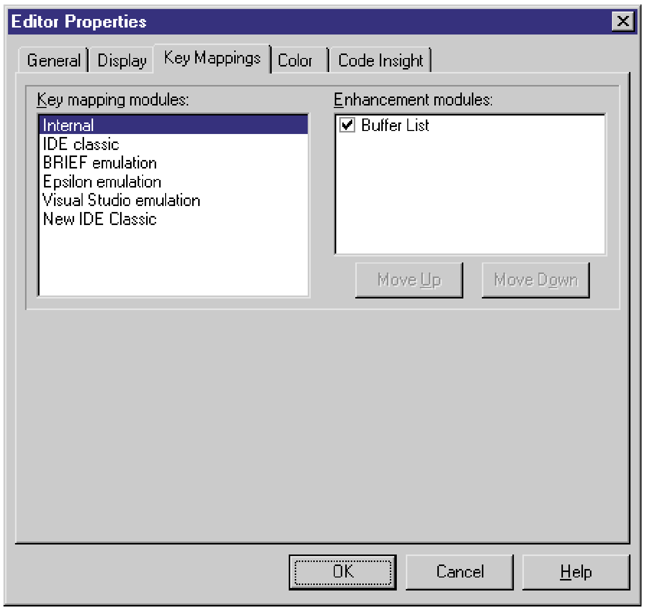


Figure 8: The Key Mappings page of the Editor Properties dialog box.

The real fun begins with the Data Diagrams page (see Figure 7), which can be used to define relationships between your various data access components, and can display this relationship visually. In Figure 7, a one-to-many relationship has been defined, represented by a line that connects CustTable to SalesTable.

Customizable key options in editor. Code Editor options are now controlled in a separate dialog box, instead of being part of the Environment Options dialog box. This new dialog box, which is displayed by selecting Tools | Editor Options, includes a new page named Key Mappings, shown in Figure 8. In addition to the previously available key mappings are the new Internal (which employs standard CUA keystroke mappings) and Visual Studio key mappings.

Delphi 5 also permits you to create custom key mappings. Doing so, however, is an involved process, employing Delphi's OpenTools API. New key mappings must be defined in code and registered with an installed design-time package using *AddKeyboardBinding*. Once a keyboard mapping has been added, it can be installed using the Key Mappings page of the Editor Properties dialog box.

See the example units in the \Demos\Key Mappings directory for more information. There you will find the source file for the Buffer List key mappings listed in the Enhancement modules list box shown in Figure 8.

Command-line options. Although not strictly an IDE feature, Delphi 5 does support a number of new command-line options. These options include the ability to load Delphi with a particular project active, suppress the splash screen, and display information about heap utilization in Delphi's title bar. Additional command-line options permit you to control debugging and compiling features. See the online Help for a full listing of the new command-line options.

TeamSource

Another major addition to Delphi 5 is TeamSource. TeamSource, which will surely be included only in the Enterprise edition, is a team project workflow tool that ensures version control and coordinates efforts of development teams. TeamSource replaces PVCS as the Delphi team development tool.

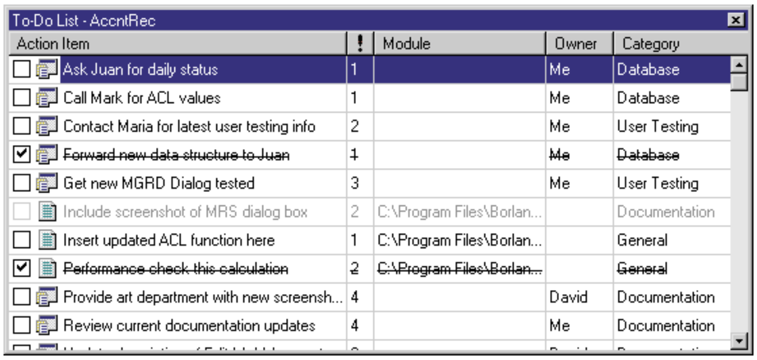
To-Do Lists

Another one of the completely new features is the To-Do List. The To-Do List permits you to identify, categorize, and manage a list of actionable items, and follow them through to their completion. These items can be either project wide, or associated with a specific unit.

Typically, project-wide To-Do items are directly added to the To-Do List dialog box, shown in Figure 9. This dialog box can be displayed by selecting View | To-Do List. Using the To-Do List right-click menu, you can add, edit, filter, sort, and otherwise manage your To-Do List.

To-Do List items can also be added directly to source code. For example, adding the following comment to a unit causes an item to be added to the To-Do List dialog box:

```
// TODO 2 -oDavid -cDocumentation  :Include screenshot of
// MRS dialog box
```



To-Do List - AccntRec

	Action Item	!	Module	Owner	Category
☐	Ask Juan for daily status	1		Me	Database
☐	Call Mark for ACL values	1		Me	Database
☐	Contact Maria for latest user testing info	2		Me	User Testing
✓	~~Forward new data structure to Juan~~	~~1~~		~~Me~~	~~Database~~
☐	Get new MGRD Dialog tested	3		Me	User Testing
☐	Include screenshot of MRS dialog box	2	C:\Program Files\Borlan...		Documentation
☐	Insert updated ACL function here	1	C:\Program Files\Borlan...		General
✓	~~Performance check this calculation~~	~~2~~	~~C:\Program Files\Borlan...~~		~~General~~
☐	Provide art department with new screensh...	4		David	Documentation
☐	Review current documentation updates	4		Me	Documentation

Figure 9: The To-Do List dialog box.

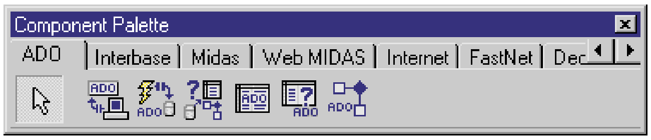


Figure 10: The ADO page of Delphi 5's Component Palette.

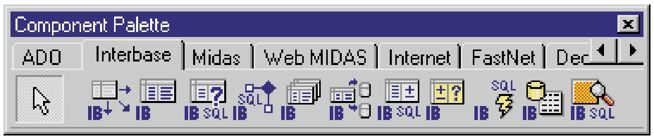


Figure 11: The Interbase page of the Component Palette.

If you subsequently use the To-Do List dialog box to mark this item completed, or edit any other element of the item, the embedded source code entry updates automatically to reflect this change in status.

New Frame Containers

The Forms unit has a new visual container class: *TFrame*. Frames are similar to forms in that they can be created as stand-alone containers for UI elements of your applications. However, they have the additional advantage of being easily embedded within existing forms or even other frames. *TFrame*, like *TForm*, descends from *TScrollingWinControl*.

After adding a new frame to a project, you can embed it into a form using the Frames component, which appears as the first component on the Standard page of the Component Palette. This item can be seen in Figure 1.

Database Enhancements

Some of the more exciting changes in Delphi 5 are associated with database development. These enhancements include support for Microsoft's ActiveX Data Objects (ADO), a major overhaul of MIDAS, and support for connecting directly to InterBase without the use of the Borland Database Engine (BDE).

ADO consists of a set of ActiveX controls that connect to data using Microsoft's COM-based OLE DB technology. Delphi 5 provides you with a series of components that encapsulate calls to underlying ADO components without requiring you to explicitly import their interfaces. These components are on the ADO page of the Component Palette (see Figure 10), and include ADOConnection, ADOCommand, ADODataSet, ADOTable, ADOQuery, and ADOStoredProc. These components can be used with Delphi's already rich set of data-aware controls.

Applications that make use of ADO components to access data do not necessarily require the BDE. They do, however, require that the ADO run-time environment, available for free from Microsoft, be installed. Nonetheless, Delphi 5's support for ADO provides developers with yet another way to access data using technology that is part of a developing industry standard.

As mentioned earlier, MIDAS has also undergone a dramatic transformation. These changes permit MIDAS to function effectively in a wider range of scenarios, including being embedded as an MTS (Microsoft Transaction Server) object. The largest of these changes involve making MIDAS stateless. In earlier versions of MIDAS, the provider components persisted information about the client connection. This information is now managed by the client, permitting the MIDAS server to be shut down and restarted as needed, without a loss of crucial state information. The changes to MIDAS run much deeper, however. See the article "MIDAS 3" by MIDAS guru Dan Miser on page 8 for a more detailed discussion.

The ADO and MIDAS technologies provide a mechanism for Delphi developers to distribute client applications without the BDE. If you're an InterBase developer, Delphi 5 provides you with yet another means of doing this. The Interbase page of the Component Palette, shown in Figure 11, includes a large number of components that permit you to attach directly to an InterBase server. Like the ADO components, Delphi 5's InterBase components integrate directly with Delphi's data-aware controls.

Internationalization

Developers needing to localize versions of their applications will also not be disappointed. The Resource DLL Wizard, first introduced in Delphi 4, gets additional support in the form of the International Translation Environment (ITE). This tool-set provides you with a means of managing translation resources, and even share them across multiple applications. This feature appears to be targeted at the Delphi Enterprise environment, but will reportedly be available separately for license for other Delphi 5 versions.

DFM Resources as Text

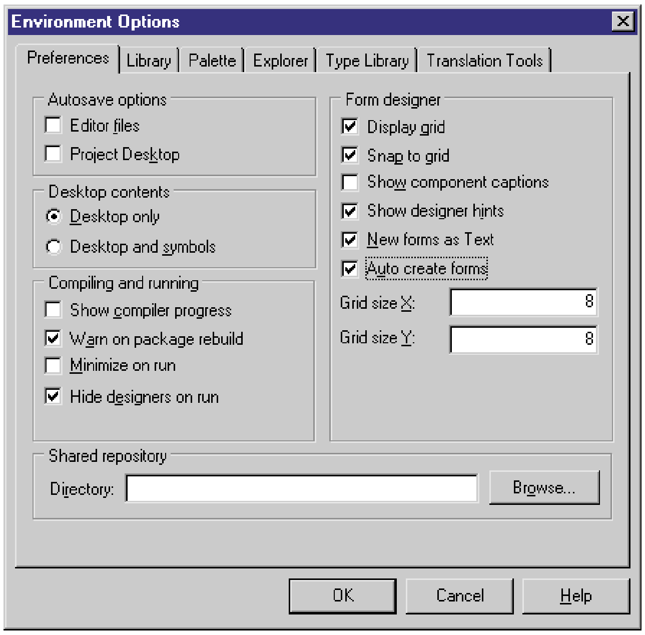
Delphi 5 now gives you control over whether forms should be saved as binary files, as they were in previous versions, or simply as text. In fact, the new default is to create forms as text, relying on the compiler to generate the DFM windows resource files before linking it into the DCU. The Preferences page of the Environment Options dialog box (see Figure 12) includes a checkbox that permits you to define whether forms are created as text by default. You can also control this by right-clicking a particular form and checking or unchecking the Text DFM option.

Control Over Auto-created Forms

Figure 12 also reveals another option that will be welcomed by a good number of Delphi developers. Delphi 5 can now to be configured to not add each and every newly created form to the auto-create list. When Auto create forms is checked, Delphi 5 acts like previous versions, auto-creating every new form. When Auto create forms is not checked, only the main form is auto-created. All other forms will appear in the Available forms list of the Project Options dialog box.

Web Component Updates

Delphi 5 includes a number of new enhancements for building Web and Internet-based applications. For one, the NetMasters Internet components have been moved to their own page on the Component Palette. No longer sharing the Internet page with the Web Broker components, the NetMasters components now appear on the FastNet page.


Environment Options

Preferences | Library | Palette | Explorer | Type Library | Translation Tools

Autosave options
- [ ] Editor files
- [ ] Project Desktop

Desktop contents
- (●) Desktop only
- ( ) Desktop and symbols

Compiling and running
- [ ] Show compiler progress
- [✓] Warn on package rebuild
- [ ] Minimize on run
- [✓] Hide designers on run

Form designer
- [✓] Display grid
- [✓] Snap to grid
- [ ] Show component captions
- [✓] Show designer hints
- [✓] New forms as Text
- [✓] Auto create forms

Grid size X: 8

Grid size Y: 8

Shared repository

Directory:

Browse...

OK | Cancel | Help

Figure 12: The Preferences page of the Environment Options dialog box.

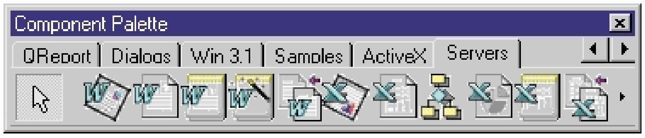


Figure 13: The Servers page of the Component Palette.

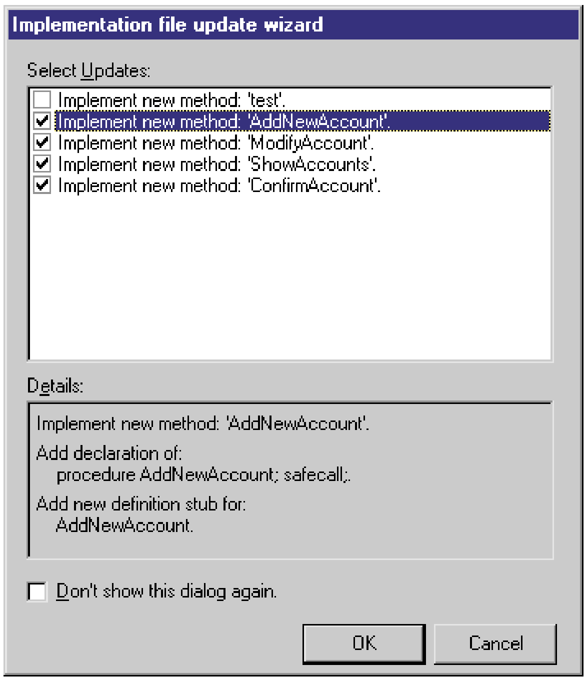
HTML support has also been updated. The NetManage HTML ActiveX control no longer appears on the Component Palette (although it can be installed if you need it for backward compatibility). Instead, Delphi 5 provides the WebBrowser component on the Internet page. This component makes use of the Microsoft Internet Explorer ActiveX control.

For developers using Active Server Pages (ASP), Delphi 5 includes a new Active Server Object Wizard. This wizard creates a Component Object Model (COM) server that can be used to define dynamic content for a Web server, in much the same way the Web Broker components do. These automation objects can be invoked from ASP pages loaded from Microsoft's Internet Information Server.

COM Enhancements

Speaking of COM, Delphi 5 includes a number of features that simplify and improve support for the standard. The most obvious of these is Delphi's support for COM servers as components. Specifically, a COM server can be imported as a component, making its use within your applications much easier. Plus, a number of standard COM servers have already been imported for your convenience. These appear on the Servers page of the Component Palette, and include Word, Excel, PowerPoint, and Outlook, to name a few (see Figure 13).

Another major update to Delphi 5's support for COM is the Implementation file update wizard (see Figure 14). This dialog box is displayed after you make changes to the type library of a COM server using the Type Library Editor. You use it to control the updates that will be automatically applied to the CoClass, i.e. the class that implements your interface.


Implementation file update wizard

Select Updates:
- [ ] Implement new method: 'test'.
- [✓] Implement new method: 'AddNewAccount'.
- [✓] Implement new method: 'ModifyAccount'.
- [✓] Implement new method: 'ShowAccounts'.
- [✓] Implement new method: 'ConfirmAccount'.

Details:

Implement new method: 'AddNewAccount'.

Add declaration of:
```
    procedure AddNewAccount; safecall;.
```
Add new definition stub for:
```
    AddNewAccount.
```

- [ ] Don't show this dialog again.

OK | Cancel

Figure 14: Use this dialog box to apply updates.

Also, Delphi now supports the sparse vtables generated for COM servers by Visual Basic (VB). When you import a VB-created COM server, Delphi automatically detects missing entries in the interfaces and generates placeholder stubs for the missing methods.

Other Enhancements

Finally, there are a number of updates that don't fall into these categories. For example, Delphi 5 includes several new wizards in its Object Repository. One of these, the Control Applet Wizard, allows you to easily create applets that appear on the Windows Control Panel. Another, the Console Wizard, permits you to create simple console applications.

There have also been a number of enhancements to Delphi's already powerful debugger. For example, in addition to the CPU window added in Delphi 4, Delphi 5 also includes an FPU window. The FPU window displays the content of the floating point processor. In addition, Delphi's debugger can be attached to an already running process. (Again, see Robert Vivrette's article "Delphi 5 Drill-Down" for more details.)

Conclusion

Delphi 5 has everything that Delphi 4 has, and more. It includes an impressive array of new features and enhancements that make it a must-have for Delphi developers. In addition, it goes a long way to restoring Delphi's reputation as the most reliable Windows development environment. ∆

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*, and is an internationally respected trainer of Delphi and Java. For information about Jensen Data Systems consulting or training services, visit http://idt.net/~jdsi, or e-mail Cary at cjensen@compuserve.com.

table of CONTENTS

By Robert Vivrette

Delphi 5 Drill-Down

Object Inspector and Debugging Enhancements

Delphi 5 should be out by the time you read this — or at least pretty close. This latest version adds all sorts of new capabilities. This article will look at a few enhancements — specifically, improvements in the Object Inspector and new capabilities of Delphi's integrated debugger.

Object Inspector Enhancements

The Object Inspector has undergone a significant facelift in capability and appearance in Delphi 5. Of course, end users won't see or appreciate these enhancements, but the developer certainly will. The main changes fall into two groups: property categories and owner-draw support.

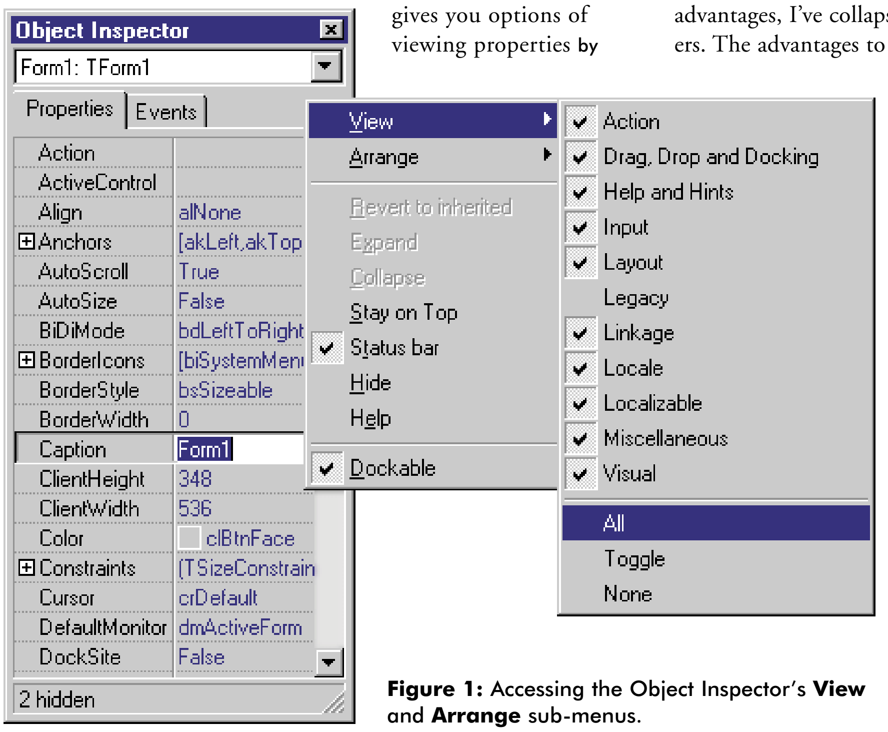
Property categories. Over the past few versions of Delphi, standard components in the VCL have continued to gain properties, such as action lists, anchors, constraints, multiple monitor support, and docking. For example, the *TForm* class has gained 16 new properties and 14 new events since Delphi 1. These additions are great; however, their appearance fills up the Object Inspector and requires a bit more hunting to find the right property or event.

Inprise's solution to this is the spiffy new property categories feature. In a nutshell, this is a way of filtering properties you want to see in the Object Inspector. Properties fall into one or more of the following categories:

- Action
- Data
- Database
- Drag, Drop, and Docking
- Help and Hints
- Layout
- Legacy
- Linkage
- Locale
- Localizable
- Miscellaneous
- Visual
- Input

When you right-click in the Object Inspector, two sub-menus become available (see Figure 1). The View sub-menu allows you to select which property categories will be shown in the Object Inspector. It also includes menu items that allow you to select all or none of the properties, as well as to toggle the currently selected properties, i.e. those that were checked are now unchecked; those that were unchecked are now checked. And just so you don't think you're looking at all the properties when some are hidden from your view, there is an optional status bar at the bottom, indicating the number of properties that are being filtered out of view.

The **Arrange** sub-menu gives you options of viewing properties **by Name** or **by Category**. The default view is by name, which is the style that we're all used to (see Figure 2).



Object Inspector

Form1: TForm1

Properties | Events

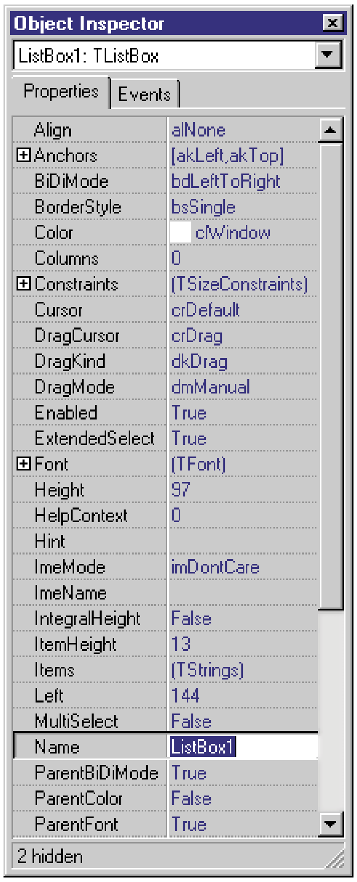
Action	
ActiveControl	
Align	alNone
⊞Anchors	[akLeft,akTop
AutoScroll	True
AutoSize	False
BiDiMode	bdLeftToRight
⊞BorderIcons	[biSystemMen
BorderStyle	bsSizeable
BorderWidth	0
Caption	Form1
ClientHeight	348
ClientWidth	536
Color	clBtnFace
⊞Constraints	(TSizeConstrain
Cursor	crDefault
DefaultMonitor	dmActiveForm
DockSite	False

2 hidden

Figure 1: Accessing the Object Inspector's **View** and **Arrange** sub-menus.

Viewing the properties by category produces a completely different look for the list (see Figure 3). To illustrate the organizational advantages, I've collapsed some of the nodes and opened up others. The advantages to this layout are subtle, but quite handy once you figure out when to use them. For example, when you're working on the design of a form, simply turn off all categories and select the **Visual** category only, which limits the properties shown to only those relating to its appearance and/or location on a form. When you're dealing with the database side of an application, you can show only database-related properties, and so on.

An interesting side effect to all this is the fact that properties can reside in more than one category. For example, the **Layout** and **Visual** categories have a number of properties in common, such as *Left*, *Top*, *Width*, *Height*, and *Align*. As you would expect, changing a property in one category instantly shows its new value in any other category in which it appears. Personally, I love the new look of the category style for the Object Inspector. Organizing the properties by function, rather than alphabetically, makes them much easier to work with.



Object Inspector

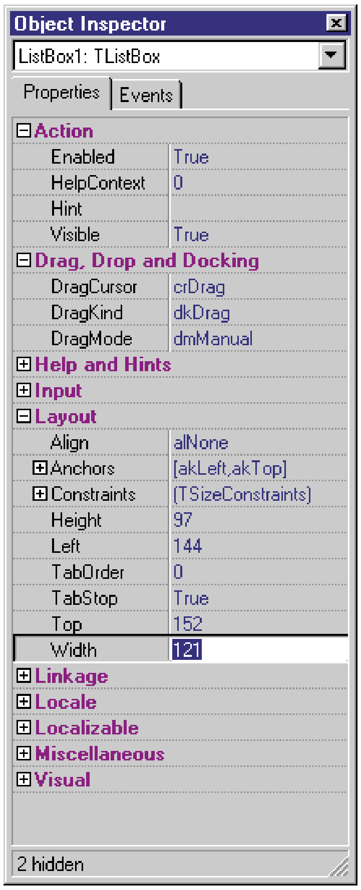
ListBox1: TListBox

Properties | Events

Align	alNone
⊞Anchors	[akLeft,akTop]
BiDiMode	bdLeftToRight
BorderStyle	bsSingle
Color	clWindow
Columns	0
⊞Constraints	(TSizeConstraints)
Cursor	crDefault
DragCursor	crDrag
DragKind	dkDrag
DragMode	dmManual
Enabled	True
ExtendedSelect	True
⊞Font	(TFont)
Height	97
HelpContext	0
Hint	
ImeMode	imDontCare
ImeName	
IntegralHeight	False
ItemHeight	13
Items	(TStrings)
Left	144
MultiSelect	False
Name	ListBox1
ParentBiDiMode	True
ParentColor	False
ParentFont	True

2 hidden

Figure 2: View by name in the Object Inspector.



Object Inspector

ListBox1: TListBox

Properties | Events

⊟**Action**	
Enabled	True
HelpContext	0
Hint	
Visible	True
⊟**Drag, Drop and Docking**	
DragCursor	crDrag
DragKind	dkDrag
DragMode	dmManual
⊞**Help and Hints**	
⊞**Input**	
⊟**Layout**	
Align	alNone
⊞Anchors	[akLeft,akTop]
⊞Constraints	(TSizeConstraints)
Height	97
Left	144
TabOrder	0
TabStop	True
Top	152
Width	121
⊞**Linkage**	
⊞**Locale**	
⊞**Localizable**	
⊞**Miscellaneous**	
⊞**Visual**	

2 hidden

Figure 3: View by category in the Object Inspector.

As a bonus, the property categories are easily available to component developers. This means components you develop can take advantage of the same capabilities by registering properties under existing property categories, or by creating entirely new categories. There is a wealth of new material supporting property categories in the DsgnIntf.pas unit.

Owner-draw support in the Object Inspector. Another enhancement to the Object Inspector is the addition of owner-draw support. In short, this means that certain properties will be able to display their values graphically rather than textually. For example, instead of seeing only *clRed* for a *Color* property, you can now also see a small swatch of the color. This capability is extended across several other properties, such as *Cursor* (see Figure 4), *Brush* and *Pen* styles, and *ImageIndex*. An important, but subtle, detail is that the *Color* property shows the real color constants currently defined on your system.

The owner-draw capabilities are a part of a revised property editor system that allow component developers to determine their own owner-draw properties. In addition to its regular value, a property now also has a "visual" value that can be represented in a way the component developer wishes. Inprise has started out hitting the big ones (as mentioned earlier), but I could see additional owner-draw properties showing up in third-party products. Just off the top of my head, I could see the *Font* property rendering the real appearance of a type face as well as glyph, picture, and icon properties showing a thumbnail of a selected image.

Debugging Enhancements

The integrated debugger in Delphi 5 has also undergone a significant overhaul, with most changes having to do with the triggering and actions taken by breakpoints. These enhancements fall into four categories: breakpoint actions, breakpoint groups, IDE command-line options, and miscellaneous enhancements.

Breakpoint actions. Before Delphi 5, breakpoints did only one thing: break out of execution while activating the debugger. Delphi 5's new breakpoint actions add a whole new set of actions that can occur when your code reaches a breakpoint:

- Break. This is pre-Delphi 5 behavior. When a break action is specified, the program halts execution and activates the debugger at that line of code.
- Log Message. This action sends a text message to the debugger's event log. Programmers can view this log by selecting View | Debug Windows | Event Log from the main menu.
- Log Expression. This is similar to the Log Message action, but allows you to specify an expression whose result will be written to the event log. This will allow you to calculate intermediate values from variables in use, and save their states in the event log.
- Eval Expression. This is the same as Log Expression, but nothing is written to the event log. The only real reason to evaluate an expression, and not to write the result, is to trigger what Inprise calls "side effects," namely the extra behavior that might occur (due to a read access method) while accessing a component property. There is also an option in the new breakpoint system that enables or disables these side effects.
- Enable Group. This action enables the specified breakpoint group.
- Disable Group. This action disables the specified breakpoint group.

Breakpoint actions are specified in the enhanced Source Breakpoint Properties dialog box (see Figure 5), which you can now access directly by right-clicking on the breakpoint glyph in the Code Editor's gutter. In Delphi 4, it was necessary to call up the Breakpoint list to access a specific breakpoint's properties. While we're on the subject, the Breakpoint list has also been revamped to include columns showing each breakpoint's assigned actions, groups, and so on.

But wait — there's more! Breakpoint actions aren't a "please pick one action" arrangement. Rather, you can specify any or all actions to occur when a breakpoint is reached. This will allow you to perform any number of the following when a breakpoint is reached:

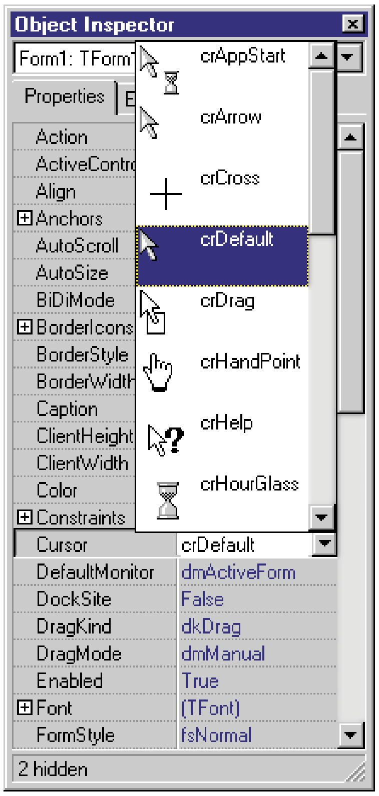
- Stop execution and activate the debugger
- Log a message indicating where you are
- Write out the result of a particular variable
- Activate a group of additional breakpoints

As you can see, you can get fairly sophisticated behavior out of the new system.

Breakpoint groups. As mentioned earlier, breakpoints can be assigned to user-defined groups for specific purposes. These groups can then be enabled or disabled, either manually from within the IDE, or programmatically as other breakpoints are encountered.

This adds a powerful new ability to debugging. Before this addition, developers had only two choices: remove breakpoints that were no longer needed, or disable each one manually by visiting each in the editor or the Breakpoint list. Developers can now establish breakpoints throughout their code and leave them disabled as a group until they're necessary. Then, by enabling the group, all these breakpoints become active.

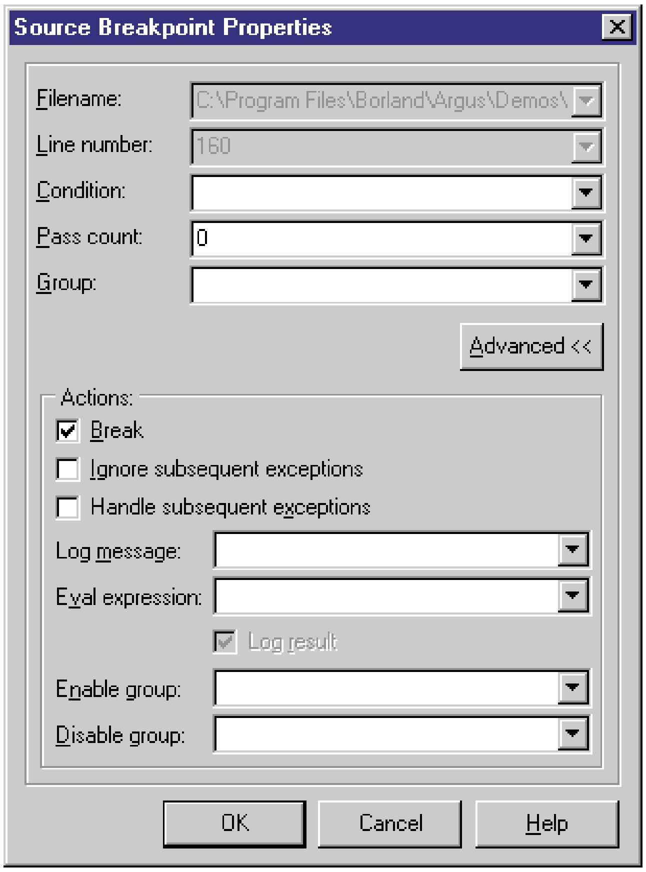
As previously mentioned, the triggering of a breakpoint can enable or disable other groups. As a result, if



Action	
ActiveContro	
Align	
⊞Anchors	
AutoScroll	
AutoSize	
BiDiMode	
⊞BorderIcons	
BorderStyle	
BorderWidth	
Caption	
ClientHeight	
ClientWidth	
Color	
⊞Constraints	
Cursor	crDefault
DefaultMonitor	dmActiveForm
DockSite	False
DragKind	dkDrag
DragMode	dmManual
Enabled	True
⊞Font	(TFont)
FormStyle	fsNormal

2 hidden

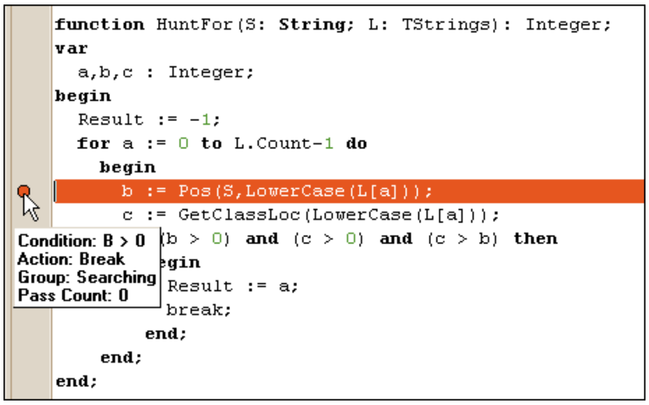
Figure 4: Now you can see the cursors (and many other things) available for use.



Filename: C:\Program Files\Borland\Argus\Demos\
Line number: 160
Condition:
Pass count: 0
Group:

[✓] Break
[ ] Ignore subsequent exceptions
[ ] Handle subsequent exceptions
Log message:
Eval expression:
[✓] Log result
Enable group:
Disable group:

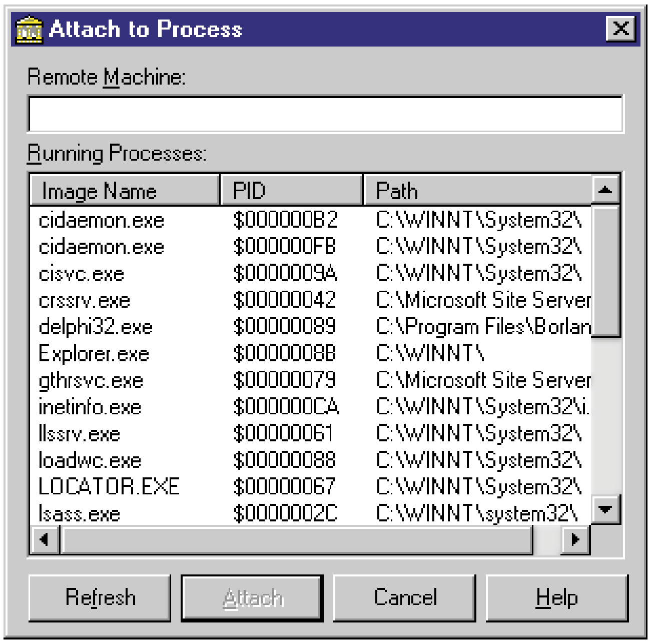
Figure 5: Specify breakpoint actions in the Source Breakpoint Properties dialog box.



```
function HuntFor(S: String; L: TStrings): Integer;
var
  a,b,c : Integer;
begin
  Result := -1;
  for a := 0 to L.Count-1 do
    begin
      b := Pos(S,LowerCase(L[a]));
      c := GetClassLoc(LowerCase(L[a]));
      if (b > 0) and (c > 0) and (c > b) then
        begin
          Result := a;
          break;
        end;
    end;
end;
```

```
Condition: B > 0
Action: Break
Group: Searching
Pass Count: 0
```

Figure 6: A tooltip, triggered by pausing the mouse over a breakpoint glyph, describing a breakpoint and its actions.



```
Image Name     PID         Path
cidaemon.exe   $000000B2   C:\WINNT\System32\
cidaemon.exe   $000000FB   C:\WINNT\System32\
cisvc.exe      $0000009A   C:\WINNT\System32\
crssrv.exe     $00000042   C:\Microsoft Site Server
delphi32.exe   $00000089   C:\Program Files\Borlan
Explorer.exe   $0000008B   C:\WINNT\
gthrsvc.exe    $00000079   C:\Microsoft Site Server
inetinfo.exe   $000000CA   C:\WINNT\System32\i
llssrv.exe     $00000061   C:\WINNT\System32\
loadwc.exe     $00000088   C:\WINNT\System32\
LOCATOR.EXE    $00000067   C:\WINNT\System32\
lsass.exe      $0000002C   C:\WINNT\system32\
```

Figure 7: A list of currently running processes.

your code detects a certain condition, it can enable breakpoint groups elsewhere.

IDE command-line options. To assist in debugging certain conditions, the IDE now supports some new command-line options. A few of the more significant ones include:

- Heap Monitor (-hm). This switch displays information in the IDE title bar regarding the amount of memory currently allocated through the memory manager.
- Heap Verify (-hv). This switch adds validation of the heap memory. Errors are displayed in the IDE's title bar.
- Auto-build (-b). This switch is used when a project or project group is also specified on the command line. In this case, the IDE starts, the project is loaded, an automatic build is performed, then the IDE exits. This is useful for doing command-line builds.
- Auto-make (-m). Same as Auto-build, but performs a standard compile (make) rather than a full build.
- OutputFile (-o<filename>). When used with the Auto-build or Auto-make option, this switch writes out Errors, Warnings, and Hints to the specified file.
- Debugger options. In addition, there are several options specifically related to the debugger. A couple of these switches make it possible to load a specified file into the debugger, attaching to a debug process by ID, Event ID, etc.

Miscellaneous debugging enhancements. In addition to breakpoint actions and groups, there is a plethora of miscellaneous improvements in the debugging area. These include:

- Breakpoint tooltips. By pausing the mouse over a breakpoint glyph in the gutter, you now get a tooltip that explains what the breakpoint is and what actions will be performed when it's triggered (see Figure 6).
- Attach to Process. A menu item under the Run menu that allows you to debug a process that is currently running, i.e. not your current process. You're presented with a list of all processes currently running, and allowed to attach to any of them (see Figure 7). Of course, it probably won't mean anything to you unless it's your own process and there's debugging information available in that process.
- Run Until Return. Also under the Run menu, this is a simple enhancement, but a real time-saver. When you're running a debugging session and you stop on a breakpoint deep within a routine, this action will cause the application to run normally until the current routine returns to its caller. In simpler terms, if you accidentally single-step into a routine and you want to get back to the line you entered from, this is the command to do it. Also, if you're content that the code is performing normally in this function and you want to immediately get back out to continue debugging, Run Until Return saves you from having to find the end of the function and using F4 to run to that point.
- Debug Inspector. Used primarily from within a watch list, this allows finer control over the inspecting of variables. You can view the registers involved, perform typecasts of the data, change the variable's value, etc.
- Process Properties. This option, accessible from the Thread Status dialog box, allows you to change temporary debugging options for a particular process.
- FPU Window. Similar to the standard CPU Window, this dialog box lets you look at the contents of the Floating Point Unit (FPU) on your computer. It gives you additional insight on MMX-specific information, FPU Registers, Control Flags, and Status Flags, to name a few.

Conclusion

As a hard-core programmer, I really appreciate it when a development tool adds features to make my job easier. I'm happy to report that Delphi 5 doesn't disappoint in this area. The enhancements to both the Object Inspector and the debugging facilities are a joy to use, and make an already great product that much better. ∆

Robert Vivrette is a contract programmer for Pacific Gas & Electric, and Technical Editor for *Delphi Informant*. He has worked as a game designer and computer consultant, and has experience in a number of programming languages. He can be reached via e-mail at RobertV@mail.com.

By Dan Miser

MIDAS 3

Delivers Robust, Efficient, Multi-tier Applications

MIDAS has grown and matured considerably since its initial release two years ago. The excitement of being able to distribute data and create thin clients was overpowering. With each release, new features have been added to make it easier to create more complex applications and solve business problems more efficiently. MIDAS 3 proves to be no exception.

Note: This article was written with a pre-release copy of Delphi 5. Features are subject to change prior to the official release of Delphi 5. Consult the manual for confirmation of the features that made it into the release version. The code for this article will be made available shortly after the release of Delphi 5 (see end of article for details).

New Events, Properties, and Components

MIDAS 3 introduces several new components, properties, and events. See the table in Figure 1 for a brief overview of these new features. However, there are more changes to MIDAS than first appear when unwrapping the package. The big changes we'll cover in this article are:

- No more *IProvider*
- No more *TProvider*
- *TSocketConnection* changes
- RDM pooling
- Threaded server changes
- Introducing *TWebConnection*
- Web MIDAS client
- Miscellaneous changes

No More *IProvider*

While new components are always great, the most significant change found in MIDAS 3 is the removal of the *IProvider* interface. *IProvider* has been central to all MIDAS applications since MIDAS first hit the streets. You would export the *IProvider* interface of a *TDBDataset* from a Remote DataModule (RDM), and the client and server applications would talk to each other using the *IProvider* interface.

Name	Description
TWebConnection	Allows MIDAS traffic to be sent using HTTP.
TXMLBroker	Interfaces between a MIDAS application server and MidasPageProducer.
TMidasPageProducer	Provides the ability to use MIDAS application servers from HTML.
TSocketConnection. SupportCallbacks	Specifies whether or not callbacks are supported for *TSocketConnection*.
TCustomProvider	
Exported	Identifies which providers are exported from the RDM.
OnGetTableName	Ease-of-use event to specify which table will be updated.
TCustomProvider	**Allows persistent state information to be ...**
AfterApplyUpdates	sent to the client.
AfterExecute	sent to the client.
AfterGetParams	sent to the client.
AfterGetRecords	sent to the client.
AfterRowRequest	sent to the client.
BeforeApplyUpdates	retrieved from the client.
BeforeExecute	retrieved from the client.
BeforeGetParams	retrieved from the client.
BeforeGetRecords	retrieved from the client.
BeforeRowRequest	retrieved from the client.
TCustomProvider.Options	
poAllowMultiRecordUpdates	Permits multiple records to be updated at once.
poDisableInserts	Stops inserts from happening on the client.
poDisableEdits	Stops edits from happening on the client.
poDisableDeletes	Stops deletes from happening on the client.
poNoReset	Ignores flag for resetting provider.
poAutoRefresh	Automatically refreshes the ClientDataset after *ApplyUpdates*.
poPropogateChanges	Sends changes that occur on the application server back to the client and merges them.
poAllowCommandText	Ability to send new SQL from the client to the server.
TClientDataset	**Allows persistent state information to be ...**
AfterApplyUpdates	retrieved from the server.
AfterExecute	retrieved from the server.
AfterGetParams	retrieved from the server.
AfterGetRecords	retrieved from the server.
AfterRowRequest	retrieved from the server.
BeforeApplyUpdates	passed to the server.
BeforeExecute	passed to the server.
BeforeGetParams	passed to the server.
BeforeGetRecords	passed to the server.
BeforeRowRequest	passed to the server.

Figure 1: New components, properties, and events found in MIDAS 3.

While this was a good model, it introduced the overhead of "state" in your application. A stateful application is one that remembers something about the previous call. For example, if you set `ClientDataset.PacketRecords=5`, the server needed to keep track of how many records have been sent to each client, introducing a stateful application. In contrast, stateless programming is on everyone's mind these days. With Microsoft preaching stateless programming as a way of life for MTS components, it's bound to get even more exposure.

So, with *IProvider* out of the picture, how do client and server communicate? Simple. We'll use the *IAppServer* interface instead. Looking at the listing in Figure 2, you'll notice many similarities between *IProvider* and *IAppServer*. The big difference is that *IAppServer* reduces the number of round-trip calls required to complete any given task, and it has no state information. You can still maintain state information on the client, but the server will be stateless. This is key for environments such as MTS that all but require stateless programming.

Practically speaking, this means that instead of right-clicking a *TDatasetProvider* component and selecting **Export Provider from Data Module**, you set `TDatasetProvider.Exported=True` for any DatasetProvider you wish to export. (More information on why I mention *TDatasetProvider* will follow shortly.) This also means you must bind a *TDataset* to a *TDatasetProvider* and export the *TDatasetProvider* in MIDAS 3. While this is different from past versions, it's a very minor price to pay, given the increased flexibility a *TDatasetProvider* component gives you as opposed to directly exporting a *TDBDataset*. The RDM's *IAppServer* interface is then used to scan the list of providers that are exported in this manner. So there are no visible changes on the client side to forge the link. You still set *ClientDataset.RemoteServer* and *ClientDataset.ProviderName* as you always have.

The down side of the new approach is that you must convert all your client applications to get away from *IProvider* and use the new *IAppServer* interface instead. This means calls to

```
type
  IAppServer = interface(IDispatch)
    ['{ 1AEFCC20-7A24-11D2-98B0-C69BEB4B5B6D }']
  function AS_ApplyUpdates(const ProviderName: WideString;
    Delta: OleVariant; MaxErrors: Integer;
    out ErrorCount: Integer; var OwnerData: OleVariant):
    OleVariant; safecall;
  function AS_GetRecords(const ProviderName: WideString;
    Count: Integer; out RecsOut: Integer; Options: Integer;
    const CommandText: WideString; var Params: OleVariant;
    var OwnerData: OleVariant): OleVariant; safecall;
  function AS_DataRequest(const ProviderName: WideString;
    Data: OleVariant): OleVariant; safecall;
  function AS_GetProviderNames: OleVariant; safecall;
  function AS_GetParams(const ProviderName: WideString;
    var OwnerData: OleVariant): OleVariant; safecall;
  function  AS_RowRequest(const ProviderName: WideString;
    Row: OleVariant; RequestType: Integer;
    var OwnerData: OleVariant): OleVariant; safecall;
  procedure AS_Execute(const ProviderName: WideString;
    const CommandText: WideString; var Params: OleVariant;
    var OwnerData: OleVariant); safecall;
  end;
```

Figure 2: We now use the *IAppServer* interface instead of *IProvider*.

ClientDataset.Provider.DataRequest and *ClientDataset.Provider.ApplyUpdates* — to name two — must be rewritten to access the new *ClientDataset.AppServer* property. A moderate application will take 15-30 minutes to convert once you understand what you need to do and you've done it a few times. Fortunately, the online Help is superb in telling you exactly what you need to do to make this conversion a successful one.

No More *TProvider*

In addition to removing the *IProvider* interface, the *TProvider* component has been removed. The reason for this change is the introduction of a new interface, the *IProviderSupport* interface. All the things a provider must do to be considered a MIDAS provider has been abstracted into this interface. As far as MIDAS is concerned, there is no longer a delineation between *TDataset* and *TDBDataset*. The onus is now on the *TDataset* descendant to tell MIDAS what to do when asked, as opposed to the provider trying to provide least-common-denominator functionality for all datasets. It's for this reason that *TDatasetProvider* is the new way to do things in MIDAS. *TProvider* still exists for backwards compatibility, but is not installed on the Component palette.

Any *TDataset* descendant that wants to participate as a MIDAS provider must implement the appropriate methods of *IProviderSupport*. For example, while *TDataset* is the first component in the hierarchy to implement *IProviderSupport*, it provides mainly blank method implementations. As you walk down the hierarchy to *TADODataset*, *TBDEDataset*, *TDBDataset*, and *TQuery*/*TTable*, further refinements to the implementation take place. When the Provider needs to get information or apply updates, it uses the *IProviderSupport* interface implementation for the *TDataset* to which it is bound.

TSocketConnection Changes

TSocketConnection has also undergone some surgery for this release. A new property, *SupportCallbacks*, has been added to the *TSocketConnection*. If this property is set to False, callbacks will not be supported in your application. Therefore, if you're not using callbacks, set this property to False. The advantage

```
class procedure TMyRDM.UpdateRegistry(Register: Boolean;
  const ClassID, ProgID: string);
begin
  if Register then
    begin
      inherited UpdateRegistry(Register, ClassID, ProgID);
      EnableSocketTransport(ClassID);
      EnableWebTransport(ClassID);
    end
  else
    begin
      DisableSocketTransport(ClassID);
      DisableWebTransport(ClassID);
      inherited UpdateRegistry(Register, ClassID, ProgID);
    end;
end;
```

Figure 3: A sample listing of the *UpdateRegistry* method.

of doing so means you only need Winsock 1 to deploy your client application. Win95 machines don't come with WinSock 2, so it was one more impediment to a smooth client-side deployment. In addition, performance will be better if you disable callback functionality.

Another change to *TSocketConnection* (and *TWebConnection*, which I'll explain later) is made on the security front. In the past, *TSocketConnection* allowed you to run any automation object on the server. This hole has been plugged by requiring you to override the *UpdateRegistry* method in the RDM. In this method, you can call one of several helper functions that will mark your application server as registered. See Figure 3 for a sample listing of one such *UpdateRegistry* method. All the helper functions will be explained throughout the course of this article.

The call to *EnableSocketTransport* simply places a registry entry under the application server's CLSID. This entry signifies that the application is available for running. One more option for you is the global override switch found in SCKTSRVR. If you uncheck the Connections | Registered Objects menu option, you eliminate the security check on the server.

RDM Pooling

RDMs are firmly rooted in COM. Because of this, the two main ways to create RDMs is to use DCOM's instancing options of either *ciMultiInstance* or *ciSingleInstance*. *ciMultiInstance* RDMs spawn one process for all clients to use. Each client will have their own RDM, but because the application is a single-threaded application, only one RDM can be working at a time. This leads to unacceptable delays in a multi-user setting.

ciSingleInstance RDMs will spawn one process per client, thus taking more memory and resources. This option is only viable if you have a few clients connecting. The reason for this is that if you're using the BDE, each process counts as one application that uses the BDE. The BDE has a limitation of 48 processes per machine. Therefore, the 49th client will not be able to spawn the 49th copy of the server.

The solution to these two sub-optimal choices prior to Delphi 5 was to use the *TThreadedClassFactory* to get the best of both worlds. One process would be spawned for all clients to share, avoiding the multiple-process problem of *ciSingleInstance*. Furthermore, each client would get its own RDM in a separate thread, thus avoiding the delay problems inherent in *ciMultiInstance*. Unfortunately, spawning unlimited threads could cause performance problems.

Delphi 5 solves all these problems by introducing object pooling for the RDM. To set up RDM pooling, you simply make a call to the *RegisterPooled* helper function in the overridden *UpdateRegistry* method of the RDM. You specify the maximum number of RDMs to create, and the amount of idle time before the RDM gets freed. If you're using free threading in your RDM, you can also mark this RDM as a singleton. The declaration for *RegisterPooled* looks like this:

```
procedure RegisterPooled(const ClassID: string;
  Max, Timeout: Integer; Singleton: Boolean = False);
```

When you attempt to instantiate an RDM that is marked for pooling from the client, an intermediary gives out one of the free RDMs in the pool. If all RDMs are in use, you'll get a "Server too busy" error message. No further work on your part is required to take advantage of RDM pooling.

At the time of this writing, RDM pooling was only available with *TWebConnection*. By the time Delphi 5 hits the shelves, *TSocketConnection* may support this feature as well.

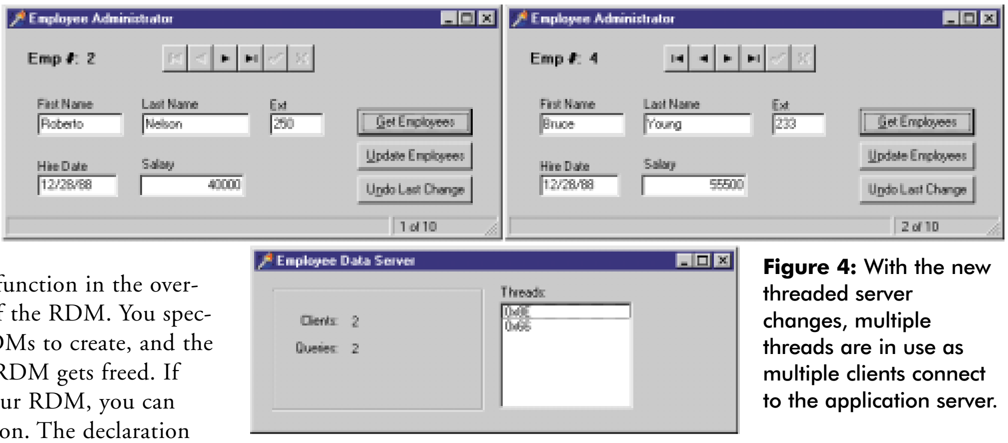
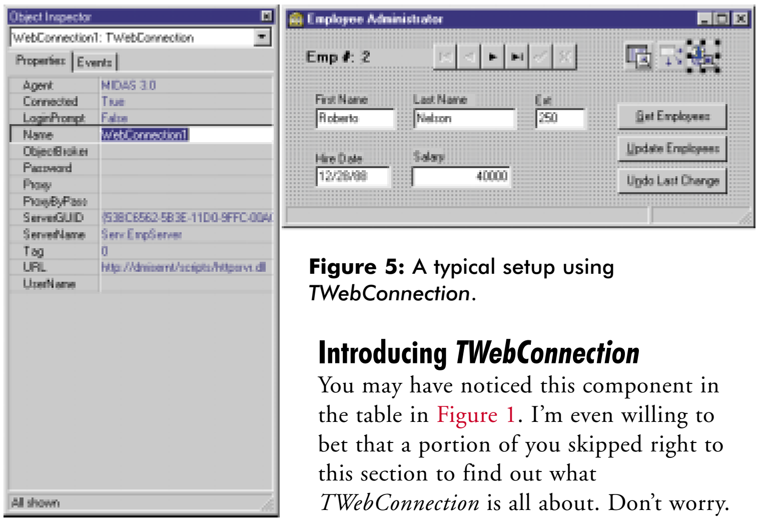


Figure 4: With the new threaded server changes, multiple threads are in use as multiple clients connect to the application server.



WebConnection1: TWebConnection

```
Agent         MIDAS 3.0
Connected     True
LoginPrompt   False
Name          WebConnection1
ObjectBroker
Password
Proxy
ProxyByPass
ServerGUID    {538C6562-5B3E-11D0-9FFC-00A
ServerName    Serv.EmpServer
Tag           0
URL           http://dmisoent/scripts/httpsrvr.dll
UserName
```

Figure 5: A typical setup using *TWebConnection*.

Threaded Server Changes

Writing multi-threaded MIDAS application servers has been pretty easy since the introduction of *TThreadedClassFactory*. This class functions as a replacement to the standard *TComponentFactory* that Delphi surfaces for COM object creation. However, *TThreadedClassFactory* has not made it through a full QA cycle, and has been relegated to the DEMOS area of the product. There have even been some reports that this class factory replacement doesn't work as well as it should when using multiple processors.

With MIDAS 3, writing a multi-threaded EXE is as easy as setting the *CoInitFlags* variable to COINIT_APARTMENTTHREADED. To make it even easier, the *TComponentFactory* now respects the *ThreadingModel* flag for EXEs, and will set *CoInitFlags* appropriately for you. For example, the following code in the **initialization** section of your RDM makes an application server multi-threaded:

```
initialization
  TComponentFactory.Create(ComServer, TEmpServer,
    Class_EmpServer, ciMultiInstance, tmApartment);
```

Remember that if you set this flag to make your application server multi-threaded, you must take care to protect global variables and GUI updates with thread-protection devices, such as Critical Sections or PostMessages. See the accompanying source code for the complete example of a multi-threaded server (see end of article for download details).

Figure 4 shows the result of this change: multiple threads are in use as multiple clients connect to the application server.

Introducing *TWebConnection*

You may have noticed this component in the table in Figure 1. I'm even willing to bet that a portion of you skipped right to this section to find out what *TWebConnection* is all about. Don't worry. You won't be disappointed.

TWebConnection is a *TSocketConnection* descendant that permits MIDAS traffic to be bundled into valid HTTP traffic, and thus use the most open port in the world, the HTTP port (default port 80). Actually, the component even supports SSL, so you can have secure communications. By doing this, all firewall issues are completely eliminated. After all, if a corporation doesn't allow HTTP traffic in or out, there is nothing that can be done to communicate with them anyway.

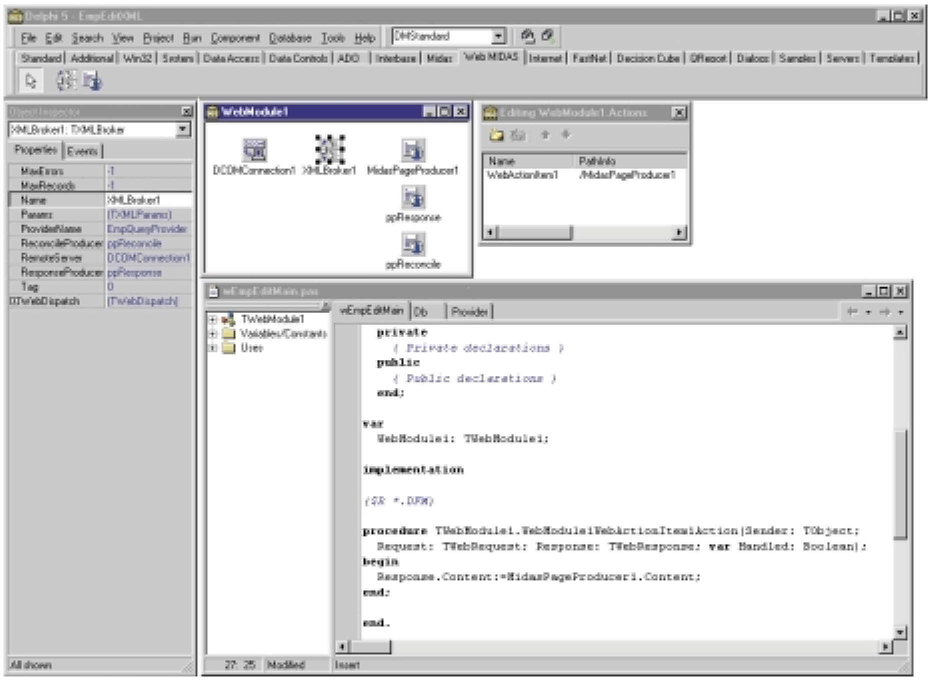
This bit of magic is accomplished by providing an ISAPI extension that translates HTTP traffic into MIDAS traffic, and vice-versa. In this regard, the ISAPI DLL does the same work that scktsrvr does for socket connections. The ISAPI extension httpsrvr.dll needs to be placed in a directory capable of executing code. For example, with IIS4, the default location for this file would be in c:\inetpub\scripts. See Figure 5 for a screen shot of a typical setup using *TWebConnection*.

Another benefit of using HTTP for your transport is that an operating system like Windows NT Enterprise allows you to cluster servers. This provides true load balancing and fault tolerance for your application server. For more information about clustering, see http://www.microsoft.com/ntserver/ntserverenterprise/exec/overview/clustering.

The limitations of using *TWebConnection* are fairly trivial, and well worth any concession in order to have more clients capable of reaching your application server. The limitations are that you must install wininet.dll on the client, and no callbacks are available when using *TWebConnection*. In addition, you must register the application server with the utility function *EnableWebTransport* in an overridden *UpdateRegistry* method. Refer back to Figure 3 for a sample listing of this method.

Web MIDAS Client

One of the more innovative features in Delphi 5 is the Web MIDAS client. Using the components found on the WebMidas tab of the Component palette, you can create a browser-only front end to your MIDAS application servers. No requirements; nothing but a Web browser hitting a Web server. This bit of magic is accomplished by using a few complementary technologies: MIDAS, XML, JavaScript, and the Delphi WebBroker.


Delphi 5 - EmpEditXML
File Edit Search View Project Run Component Database Tools Help
DHStandard
Standard Additional Win32 System Data Access Data Controls ADO Interbase Midas Web MIDAS Internet FastNet Decision Cube QReport Dialogs Samples Servers Templates
Object Inspector
Properties Events
WebModule1
DCOMConnection1
XMLBroker1
MidasPageProducer1
ppResponse
ppReconcile
Editing WebModule1.Actions
vEmpEditMain.pas
TWebModule1
Variables/Constants
Uses
vEmpEditMain
Db
Provider
All shown
27: 25
Modified
Insert

XMLBroker1: TXMLBroker

MaxErrors	-1
MaxRecords	-1
Name	XMLBroker1
Params	(TXMLParams)
ProviderName	EmpQueryProvider
ReconcileProducer	ppReconcile
RemoteServer	DCOMConnection1
ResponseProducer	ppResponse
Tag	0
OTWebDispatch	(TWebDispatch)

Name	PathInfo
WebActionItem1	/MidasPageProducer1

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  WebModule1: TWebModule1;

implementation

{$R *.DFM}

procedure TWebModule1.WebModule1WebActionItem1Action(Sender: TObject;
  Request: TWebRequest; Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content:=MidasPageProducer1.Content;
end;

end.
```

Figure 6: Some of the key components and settings for the Web MIDAS client at design time.

To build a complete Web client, you must understand the WebBroker architecture. This article will not explore this topic in any detail, but rather provide an overview of the steps you need to take to build a rudimentary Web MIDAS client.

First, you need to create a Web server application by selecting **File | New | WebServer**. I'll create an ISAPI application here, as the application will run under IIS4. Next, add a *TDCOMConnection* component to the WebModule and set the *ServerName* property. This will act as your conduit to the application server from the ISAPI application.

Next, place a *TXMLBroker* component on the WebModule and link it to the *DCOMConnection* component you just placed on the WebModule. In this regard, you can think of the *XMLBroker* component as the WebMidas equivalent of the *TClientDataset*. The main difference is that the *XMLBroker* uses XML data packets instead of the OleVariants used by ClientDatasets.

If you want to specify an HTML page that will be sent automatically after a successful *ApplyUpdates*, you can tie the *XMLBroker.ResponseProducer* to any *PageProducer* component. Reconciliation errors will be dealt with by specifying a *PageProducer* capable of dealing with these errors. A standard HTML page, ERR.HTML, is included in the <Delphi>\SOURCE\WEBMIDAS directory. It serves as a convenient starting place to deal with reconciliation errors, much like the RECERROR.PAS unit is the standard way to deal with reconciliation errors in Windows applications. We'll ignore the other properties and events on the *XMLBroker* component.

Finally, place a *MidasPageProducer* component on the WebModule. This is the key component that will generate the HTML content that the browser will eventually see. Notice that the *HTMLDoc* property has a default template inside it already. The HTML tags will expand at design time, when you create the HTML content by using the Web Page Editor (available by double-clicking on *TMidasPageProducer)*.

The main property to note on *TMidasPageProducer* is *IncludePathURL*. This is the property that will determine where the application will look for the JavaScript files that help produce the WebMidas application. You'll need to deploy the *.js files found in <Delphi>\SOURCE\WEBMIDAS to a location accessible by your ISAPI application.

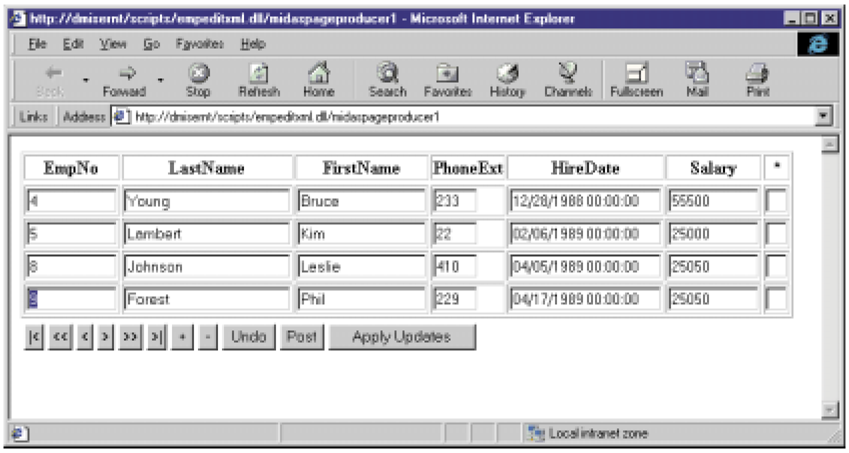
To customize the HTML that will display from your ISAPI application, press the **New** button in the Web Page Editor. For this sample, we'll choose **DataForm**. While selecting DataForm in the tree view, press **New** again. You'll be presented with a list of components that you can add based on your selection. Select **DataGrid** and **DataNavigator** to add these elements to your HTML page.

At this point, select the DataGrid element in the tree view and view the Object Inspector. If you set the *XMLBroker* property to the component you added above, you'll see a grid display in the resulting HTML code below. Do the same thing for the DataNavigator and you'll see a group of buttons that look like the DBNavigator. You can customize just about any attribute or element on the HTML page.

Figure 6 shows some of the key components and settings at design time.

Now that we've built the page, we need to send the resulting HTML page to a Web browser. The content is generated by creating a new WebModule Action (by double-clicking on the WebModule and pressing the **Add** button) and assigning the *OnAction* event to the following code:

```
procedure TWebModule1.WebModule1WebActionItem1Action(
  Sender: TObject; Request: TWebRequest;
  Response: TWebResponse; var Handled: Boolean);
begin
  Response.Content:=MidasPageProducer1.Content;
end;
```


http://dmisernt/scripts/empeditxml.dll/midaspageproducer1 - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Links Address http://dmisernt/scripts/empeditxml.dll/midaspageproducer1
|< << < > >> >| + - Undo Post Apply Updates
Local intranet zone

EmpNo	LastName	FirstName	PhoneExt	HireDate	Salary	*
4	Young	Bruce	233	12/28/1988 00:00:00	55500	
5	Lambert	Kim	22	02/06/1989 00:00:00	25000	
8	Johnson	Leslie	410	04/05/1989 00:00:00	25050	
9	Forest	Phil	229	04/17/1989 00:00:00	25050	

Figure 7: The Web MIDAS client in action.

Remember to set the *PathInfo* of the *Action* to something meaningful. This value will be used from the browser later to access the virtual page you just created.

To deploy this application, you need to compile the application and copy the resulting ISAPI DLL to a place that is capable of running scripting code, e.g. c:\inetpub\scripts. You also need to place your JavaScript files in the path mentioned in *TMidasPageProducer.IncludePathURL*. The only thing left is to fire up your browser and connect to the page, either directly or via a link on another HTML page. For example, typing:

```
http://dmisernt/scripts/empservxml.dll/MidasPageProducer1
```

would work if you specified:

```
TWebModule.Actions[0].PathInfo=MidasPageProducer1.
```

Figure 7 shows the Web MIDAS client in action.

Miscellaneous Changes

Deployment of MIDAS applications has changed slightly with this release. Because *IProvider* is gone, there is no more need to deploy STDVCLnn.DLL. Also, DBCLIENT.DLL got a name change to MIDAS.DLL. Furthermore, COM is not being used to communicate with MIDAS.DLL. It's still registered with COM, and COM objects exist inside it, but the mechanism to retrieve those objects makes calls to *CoCreateInstance* unnecessary. The real benefit to the new approach is that you should see no more "CoInitialize has not been called" errors.

One of the primary benefits of writing a multi-tier application is the ability to partition your business logic into the middle tier. In this tier, you can evaluate data as it comes from the client and is about to be written to the server. You may even end up changing some of the values in this tier to help the data conform to your business rules. Until MIDAS 3, you needed to devise your own mechanism to update the record, or refresh the entire ClientDataset — which could prove to be a very costly operation.

With the addition of the *Provider.Options.poPropogateChanges* property, any change you make on the application server will automatically be returned to the client and merged into the ClientDataset. Another common reason for using this feature will be when you use auto-incrementing keys, date-time stamps, or triggers to modify record values on the server.

The *Provider.Options* set added many other new values as well. Some of the more intriguing ones are the *poDisableInserts*, *poDisableEdits*, and *poDisableDeletes* elements. By setting these properties appropriately, you can prevent the ClientDataset that is linked to this *TProvider* from inserting, editing, or deleting records. This provides a new opportunity to centralize more business logic and security implementation details on the server, which is always a good thing when trying to make your client as thin as possible.

The ability to send dynamic SQL to the server is easier than ever. Using the *ClientDataset.CommandText* property, you can send a SQL statement from the client to the server. As long as you set *Provider.Options.poAllowCommandText* to True, this change will be accepted. The statement will be executed when you issue either an *Open* or *Execute* command. *Execute* is a new method that allows you to execute queries on the server that don't generate a result set. Output parameters will be passed back, but no result set. This brings ClientDataset more in line with *TQuery* and *TStoredProc*.

Lastly, I would be remiss not to point out that this version of MIDAS appears to be far more stable than its predecessor, even while in pre-release. Bugs that were present in MIDAS 2 have been dealt with. And new features and functionality appear to have a very low defect rate. This optimal combination of new features and reduced bug count makes MIDAS 3 very impressive.

Conclusion

This article covered many of the new features that MIDAS 3 introduced. I'm sure there will be features in the release version that were not available in the version used to do this review. However, this is only the beginning. Each topic is begging for your detailed exploration to find out how you can best use MIDAS to create robust, efficient multi-tier applications. MIDAS has always been able to deliver data easily, and MIDAS 3 does just that — it delivers. ∆

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\99\AUG\DI9908DM.

Dan Miser is a long-time Delphi programmer and consultant, specializing in multi-tier application design using MIDAS. He is active in the Borland newsgroups, where he serves as a proud member of TeamB (http://www.teamb.com). Dan also finds time to write for *Delphi Informant* and speak at Borland conferences. You can visit his Web site at http://www.execpc.com/~dmiser, or contact him at dmiser@execpc.com.

table of CONTENTS

By Ashley Davy

Network Share

Getting Information about Network Resources

With distributed computing on the rise, obtaining network information is often a necessity for the Windows application developer. For example, a developer might want to locate all the available printers on the network, connect to a remote registry, search for a particular DCOM object, or simply connect to a network share. All these tasks involve finding and identifying network resources.

Accomplishing these tasks can seem daunting at first since Delphi doesn't include a component or functions to return resources on the network. You must therefore turn to the Windows API, a set of library DLLs at the heart of the Windows operating system. There are some differences between API calls for the different Windows environments. Some API calls available for one operating system may not be available or work properly with other Windows operating systems. However, all calls listed here will work for the Windows 95, 98, and NT platforms.

Delphi includes these libraries in its native units and provides wrapper functions and procedures for most of these API calls. The DLL containing the function calls that we'll indirectly reference is mpr.dll. These functions are then prototyped in Windows.pas, which will be included in our application. The Win32 Programmer's Reference is a Help file that comes with Delphi. (If you can't locate it on your machine, you may not have loaded it when you installed Delphi.) It contains a wealth of information and insight into Windows programming.

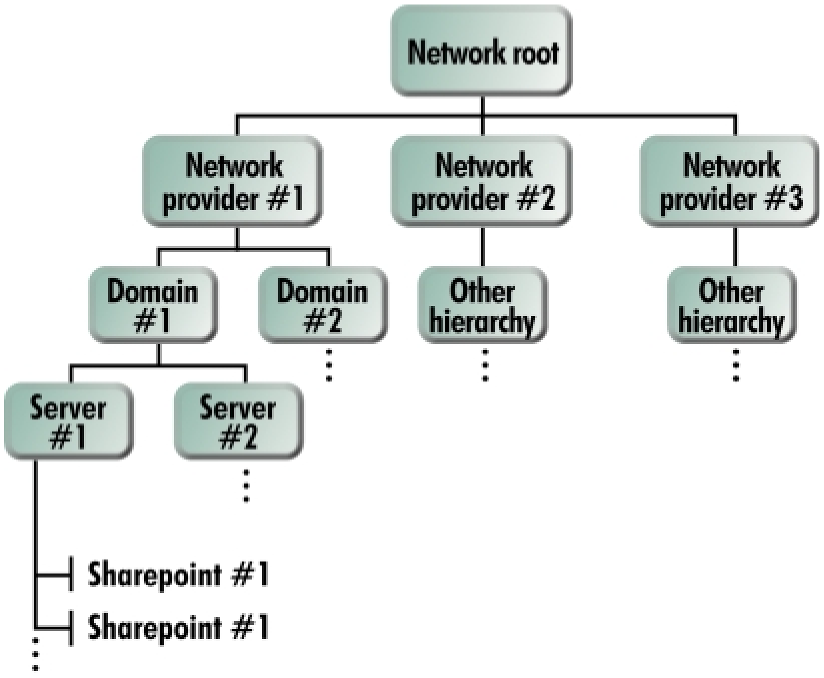


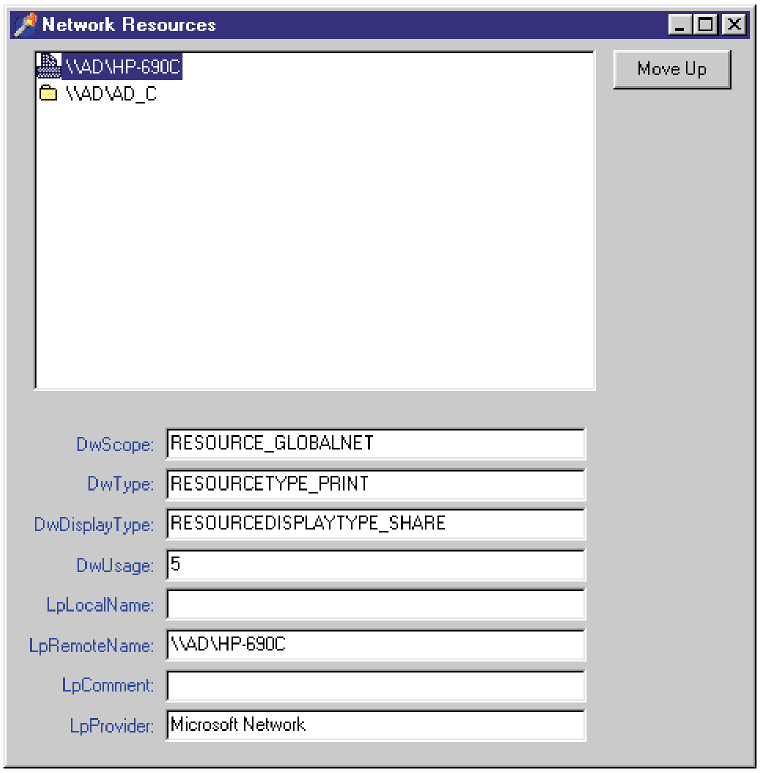
Figure 1: Hierarchical organization of a network (from the Win32 Programmer's Reference Help file).

Understanding the Hierarchy

The networking environment is organized into a hierarchy (see Figure 1). There are basically two types of objects or data structures that describe a particular resource: a container, and a connectable resource. A container resource would be a particular domain or server, whereas a connectable resource would be a printer or shared directory. Each container may have connectable and other container objects. With this in mind, only container resources can be enumerated because a connectable resource can't have any other network information. The network hierarchy is formed through these container resources. In obtaining network information, it's not necessary to have knowledge of the particular computer network in question. To enumerate all the resources on the network, it's only necessary to start at the network root because it's organized in a hierarchical fashion.

Once the network root resource is obtained (which is a container, by the way), this resource can then be enumerated to obtain all its container and/or connectable resources. After this has been accomplished, each resource that was obtained from the network root can then be tested to see if it's a container resource. If it is, it can then be enumerated. This process will repeat until all the resources on the network have been enumerated.

Looking at the program, its function is similar to network neighborhood (see Figure 2). When the program starts, it opens with the network root in the list view. Double-clicking on a particular listing will cause that resource to be enumerated. Clicking on an item will display that particular resource's information in the bottom part of the form. The **Move Up** button will move up to the previously enumerated resource. For example, if you're on a server resource and double-click, two directory shares are displayed. At this point, if the **Move Up** button is clicked, the previously enumerated resource (the server in this case) will be displayed.



DwScope: RESOURCE_GLOBALNET
DwType: RESOURCETYPE_PRINT
DwDisplayType: RESOURCEDISPLAYTYPE_SHARE
DwUsage: 5
LpLocalName:
LpRemoteName: \\AD\HP-690C
LpComment:
LpProvider: Microsoft Network

Figure 2: The Network Resources dialog box.

Examining the Code

Working with the Windows API usually isn't straightforward, but we have little choice. Using the API is usually difficult for three reasons:

- It's hard to locate the proper function or functions to use to accomplish a specific task.
- Once you've located the functions, they're poorly documented as to their implementation.
- The functions are documented in reference to C code, making it even more challenging for Pascal programmers.

For this program, the Windows API calls that will be used are *WNetOpenEnum*, *WNetEnumResource*, and *WNetCloseEnum* — three simple functions with many parameters, and even more pitfalls.

The procedure at the heart of the application is *GetNetRes*, which is located in the *UNetResources* unit, shown in Listing One (beginning on page 23). When calling this procedure, two parameters are passed in, of types *TStringList* and *TNetResA*. The *TStringList* is created before being passed into the procedure. The *TNetResA* class type mimics the *TNetResource* record structure defined in the Windows.pas unit (see Figure 3).

```
// Used to load data into/from the NetResourceA record.
type
  PPNetResA = ^TNetResA;
  TNetResA = class(TObject)
  public
    dwScope: Cardinal;
    dwType: Cardinal;
    dwDisplayType: Cardinal;
    dwUsage: Cardinal;
    lpLocalName: string;
    lpRemoteName: string;
    lpComment: string;
    lpProvider: string;
    NetResOwner: TNetResA;
  end;

// Record structure of NetResource as declared in Windows.pas.
type
  PNetResourceA = ^TNetResourceA;
  TNetResource = TNetResourceA;
  TNetResourceA = _NETRESOURCEA;
  NETRESOURCEA = _NETRESOURCEA;
  _NETRESOURCEA = packed record
    dwScope: DWORD;
    dwType: DWORD;
    dwDisplayType: DWORD;
    dwUsage: DWORD;
    lpLocalName: PAnsiChar;
    lpRemoteName: PAnsiChar;
    lpComment: PAnsiChar;
    lpProvider: PAnsiChar;
  end;
```

Figure 3: Two parameters of types *TStringList* and *TNetResA* are passed when calling the *GetNetRes* procedure (from the Windows.pas unit included with Delphi).

The following array structure is defined in the **type** section of the *GetNetRes* procedure:

```
type
  PCompResArray = ^TCompResArray;
  TCompResArray = array[0..10000] of TNetResource;
```

This is a very important part of retrieving network resource information. The first step in obtaining the network resource information is copying the data in the *TNetResA* object (referenced in the procedure as *NetResOwner*) into a declared procedural variable named *NetResA*. The variable *NetResA* is of type *NetResourceA*, which is a record. The reason for the switching of the data between the record type and the object is because it's usually easier to work with objects than with record types. In moving the data to the record, the last four data members are of type *PAnsiChar*. When populating these data members, memory will have to be allocated for each one with the *StrAlloc* function, then copied into the *PAnsiChar* with the *StrPCopy* function.

There is a special case in which no data will be copied, and that's if the *NetResOwner* is **nil**. This special case indicates the top of the network hierarchy. Once the data is copied into the record structure or the *TNetResA* object is found **nil**, then it's time to make the first API call. The first call is made using the function *WNetOpenEnum*. A usage description and a parameter listing for this call and all other API calls can be found in the Win32 Programmer's Reference Help file.

The purpose of *WNetOpenEum* is to open a handle to a specific network resource defined by the *NetResourceA* record. This function takes five parameters. The first is the scope of the enumeration. Being interested in all network resources, this parameter is set to the predefined constant RESOURCE_GLOBALNET. The next parameter describes the type of resources to enumerate. This is set to RESOURCETYPE_ANY. The third parameter is the usage parameter, and, because we're interested in obtaining all network resources, this parameter will be set to zero. The fourth parameter is a pointer to our *NetResourceA* record. In the first call to this function, this parameter will be **nil** in order to obtain the handle to the top-level container to enumerate. The last parameter is the network resource handle. The value passed in is meaningless. The variable passed into the function only has meaning when the function returns NO_ERROR. This indicates the function call was successful, and the network resource handle is stored in the address of the fifth parameter, i.e. the variable that was passed in.

With a successful call to *WNetEnumOpen* the attention can then be turned to the next API call: *WNetEnumResource*. This call is more difficult, because this function plays many different roles and has to be called repeatedly to get the enumerated information. This function takes four parameters, the first of which is the network resource handle obtained by the function call *WNetEnumOpen*. The second parameter is an integer that defines the number of entries requested. In this particular example, all entries are requested, so each time this function is called, it's set to the maximum value, which is *MaxDWord* (defined in Windows.pas). When the function is successful, the second parameter will contain the number of entries actually read.

```
// Enter loop to continually enumerate the resource until
// all resources have been acquired.
repeat
  I := 0;
  // Obtain the resources that were enumerated with the
  // first call to WNetEnumResource.
  for I := 0 to ResourceEntries - 1 do begin
    NetResB := TNetResA.Create;
    NetResB.dwScope := ComputerResources^[I].dwScope;
    NetResB.dwType := ComputerResources^[I].dwType;
    NetResB.dwDisplayType :=
      ComputerResources^[I].dwDisplayType;
    NetResB.dwUsage := ComputerResources^[I].dwUsage;
    NetResB.lpLocalName :=
      StrPas(ComputerResources^[I].lpLocalName);
    NetResB.lpRemoteName :=
      StrPas(ComputerResources^[I].lpRemoteName);
    NetResB.lpComment :=
      StrPas(ComputerResources^[I].lpComment);
    NetResB.lpProvider :=
      StrPas(ComputerResources^[I].lpProvider);
    NetResB.NetResOwner := NetResOwner;
    NetResList.AddObject(IntToStr(I) + ' - ' +
                         NetResB.lpRemoteName,NetResB);
  end;

  // Free the memory for the buffer and re-initialize
  // the variables.
  FreeMem(ComputerResources, BufferSize);
  ResourceEntries := MaxValue;
  BufferSize := 0;
  // Call WNetEnumResource again to determine the amount
  // of memory that is needed for the next enumeration of
  // network resource array information.
  ReturnValue := WNetEnumResource(NetResHand,
    ResourceEntries, ComputerResources, BufferSize);
  // Allocate the memory and re-initialize the
  // ResourceEntries variable.
  GetMem(ComputerResources, BufferSize);
  ResourceEntries := MaxValue;
  // Call WNetEnumResource again to continue the enumeration of
  // the network resource.
  ReturnValue := WNetEnumResource(NetResHand,
    ResourceEntries, ComputerResources, BufferSize);

until not((ReturnValue = NO_ERROR) or
         (ReturnValue = ERROR_MORE_DATA));
```

Figure 4: This loop moves data into objects loaded into *TStringList*, and calls *WNetEnumResource* until there are no more obtainable data.

The third and fourth parameters work together. The third is a pointer of the type *PCompResArray*. This type was defined in the *GetNetRes* procedure. The last parameter is the buffer size, which has a dual role. It specifies the memory size of the third parameter; however, if the call is made and the buffer size is zero, upon a successful return this last parameter will then contain the size of the memory block the third parameter will need.

Now the appropriate sequencing of these two API calls can be made to obtain the network resource information. The first thing to do is call *WNetOpenEnum* to get the resource handle for the particular resource in question. Having obtained the handle call *WNetEnumResource*, make sure the second parameter is set to the maximum value and the last parameter, which is the buffer size, is set to zero. This call is successful when it returns a value of NO_ERROR or ERROR_MORE_DATA.

In this step, *WNetEnumResource* isn't going to return data, but it will return the amount of memory needed for the data structure that is passed in. With a successful call, the amount of memory needed to be allocated will be contained in the last parameter. Now that the amount of memory needed for the structure is known, it can be allocated. It's important to note that in the beginning of this procedure, a type structure was defined, and the variable *ComputerResources* was declared as *PCompResArray*:

```
type
  PCompResArray = ^TCompResArray;
  TCompResArray = array[0..10000] of TNetResource;

var
  ComputerResources: PCompResArray;
```

All that has happened up to this point is that the structure of the memory has been defined. The actual space or memory block hasn't been allocated for this structure. It's simply a pointer to a list of pointers of *TNetResource* records. With this in mind, the memory can now be allocated with the following call:

```
GetMem(ComputerResoures, BufferSize);
```

where the *ComputerResources* array is the pointer to the structure of the memory block, and *BufferSize* is the size of the memory needed (returned from the first call to *WNetEnumResource*). The next step is to set the variable *ResourceEntries* back to its maximum value. These two items being completed, a second call to *WNetEnumResource* is made. In this call, there are three things to note. The first is that any data that is returned will be in the *ComputerResources* variable. The second is the *ResourceEntries* variable, which tells how many record items were returned. The third item to note is the return value of the function. If this value is NO_ERROR or ERROR_MORE_DATA, there will be subsequent calls to *WNetEnumResource* to retrieve more information.

With these items noted, we're now ready to process the information in a loop to move the data out of the records and into objects that will be loaded into the *TStringList*, which was passed into the procedure. The loop is used to accomplish two tasks: to move the data from the records into objects and continually call *WNetEnumResource* until there is no more data to be obtained (see Figure 4).

There's a **for** loop inside the **repeat** loop that will iterate through the *ComputerResources* array and de-reference each data element and load it into the object. The number of iterations depends on the *ResourceEntries* variable, which contains the number of entries returned from the call to *WNetEnumResource*.

Once the **for** loop has finished, the steps that were implemented in the beginning of the procedure are repeated with some subtle differences. First the memory that was allocated is freed, and the variables *ResourceEntries* and *BufferSize* are set to *MaxValue* and zero, respectively. *WNetEnumResource* is called again to determine the buffer size needed for the group of information to be retrieved. This value memory is allocated to the *ComputerResources* pointer, and *ReturnEntries* is set to *MaxValue*. The call is then made to *WNetEnumResource* again, this time to populate *ComputerResources* with the network resource information. This process is repeated until a return value other than NO_ERROR or ERROR_MORE_DATA is encountered.

Conclusion

Workers are becoming increasingly dependent not only on their own computers, but also on the networks to which they're connected — and located on these various networks are resources of one kind or another. As the technology of computers has developed, we've seen operating systems evolve from being able to run only one application at a time to now running many programs at once. This being the case, the next logical step was for these programs to communicate with each other. This is the same progression that is being felt in the networking environment between computer systems today. Whether it's applications communicating with each other or various other resources that need to be utilized from a remote machine, users will come to expect computer-to-computer communication. Δ

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\99\AUG\DI9908AD.

Ashley Davy is a software engineer at Medic Computer Systems in Raleigh, NC. He has been developing Windows software for approximately four years, focusing mainly in areas of memory management and operating systems programming. He also has two certifications with Microsoft: Microsoft Product Specialist and Microsoft Certified Solutions Developer (MCSD). He can be reached via e-mail at adavy@bellsouth.net.

Begin Listing One — UNetResources.pas

```
unit UNetResources;

interface

uses
  Windows, SysUtils, Dialogs, Classes;

// Class used to load data into/from
// the NetResourceA record.
type
  PPNetResA = ^TNetResA;
  TNetResA = class(TObject)
  public
    dwScope: Cardinal;
    dwType: Cardinal;
    dwDisplayType: Cardinal;
    dwUsage: Cardinal;
    lpLocalName: string;
    lpRemoteName: string;
    lpComment: string;
    lpProvider: string;
    NetResOwner: TNetResA;
  end;

// Procedure used to retrieve network resoures that a
// container network resource may contain.
procedure GetNetRes(NetResList: TStringList;
  NetResOwner: TNetResA);

var
  NetResList: TStringList;
  NResA: PNetResourceA;

implementation

procedure GetNetRes(NetResList: TStringList;
  NetResOwner: TNetResA);

type
  PCompResArray = ^TCompResArray;
  TCompResArray = array [0..10000] of TNetResource;

const MaxValue = 4294967295;

var
  I, K: Integer;
  ReturnValue: Cardinal;
  NetResHand: Cardinal;
  ResourceEntries: Cardinal;
  BufferSize: Cardinal;
  ComputerResources: PCompResArray;
  NetRes,NetResB: TNetResA;
  NetResA: NetResourceA;
  PNetResA: PNetResourceA;
  ReturnValueStr: string;

begin
  // Convert the NetResOwner, which is a TNetResA Object to
  // a PNetResourceA record. If NetResOwner is nil, this
  // indicates that the top-level network resource is being
  // enumerated.
  if Assigned(NetResOwner) then
    begin
      NetResA.dwScope := NetResOwner.dwScope;
      NetResA.dwType := NetResOwner.dwType;
      NetResA.dwDisplayType := NetResOwner.dwDisplayType;
      NetResA.dwUsage := NetResOwner.dwUsage;
      NetResA.lpLocalName :=
        StrAlloc(Length(NetResOwner.lpLocalName) + 1);
      StrPCopy(NetResA.lpLocalName,
               NetResOwner.lpLocalName);
      NetResA.lpRemoteName :=
        StrAlloc(Length(NetResOwner.lpRemoteName) + 1);
      StrPCopy(NetResA.lpRemoteName,
               NetResOwner.lpRemoteName);
      NetResA.lpComment :=
        StrAlloc(Length(NetResOwner.lpComment) + 1);
      StrPCopy(NetResA.lpComment, NetResOwner.lpComment);
      NetResA.lpProvider :=
        StrAlloc(Length(NetResOwner.lpProvider) + 1);
      StrPCopy(NetResA.lpProvider,NetResOwner.lpProvider);
      PNetResA := Addr(NetResA);
    end
  else
    PNetResA := nil;

  // Intialize Values for API call. The buffer size is the
  // amount of memory needed to store the record structures
  // returned.
  BufferSize := 0;
  ResourceEntries := MaxValue;

  // Open handle to network resource.
  ReturnValue := WNetOpenEnum(RESOURCE_GLOBALNET,
    RESOURCETYPE_ANY, 0, PNetResA,NetResHand);

  // If the return value is anything other than NO_ERROR,
// then exit the procedure.
if ReturnValue <> NO_ERROR then begin
  case ReturnValue of
    NO_ERROR:
      ReturnValueStr := 'NO_ERROR';
    ERROR_NOT_CONTAINER:
      ReturnValueStr := 'ERROR_NOT_CONTAINER';
    ERROR_INVALID_PARAMETER:
      ReturnValueStr := 'ERROR_INVALID_PARAMETER';
    ERROR_NO_NETWORK:
      ReturnValueStr := 'ERROR_NO_NETWORK';
    ERROR_EXTENDED_ERROR:
      ReturnValueStr := 'ERROR_EXTENDED_ERROR';
  end;

  MessageDlg(ReturnValueStr, mtError, [mbOk], 0);
  Exit;
end;

// This API call is used to enumerate the container
// network resource to obtain the other network resources
// that fall below this item in the network hierarchy.
// The first call to this API function is used to
// determine the size of the memory block that will be
// needed to hold the array values. The size of the
// memory block will be stored in BufferSize.
ReturnValue := WNetEnumResource(NetResHand,
  ResourceEntries, ComputerResources, BufferSize);

// If the return value is anything other than NO_ERROR or
// ERROR_MORE_DATA, then exit the procedure.
if Not((ReturnValue = NO_ERROR) or
   (ReturnValue = ERROR_MORE_DATA)) then begin
  case ReturnValue of
    NO_ERROR:
      ReturnValueStr := 'NO_ERROR';
    ERROR_NO_MORE_ITEMS:
      ReturnValueStr := 'ERROR_NO_MORE_ITEMS';
    ERROR_MORE_DATA:
      ReturnValueStr := 'ERROR_MORE_DATA';
    ERROR_INVALID_HANDLE:
      ReturnValueStr := 'ERROR_INVALID_HANDLE';
    ERROR_NO_NETWORK:
      ReturnValueStr := 'ERROR_NO_NETWORK';
    ERROR_EXTENDED_ERROR:
      ReturnValueStr := 'ERROR_EXTENDED_ERROR';
  end;

  MessageDlg(ReturnValueStr, mtError, [mbOk], 0);
  Exit;
end;

// Initialize entries value and allocate the memory for
// the enumeration buffer.
ResourceEntries := MaxValue;
GetMem(ComputerResources, BufferSize);

// Second call to EnumResources to actually load the data
// into the ComputerResources array.
ReturnValue := WNetEnumResource(NetResHand,
  ResourceEntries, ComputerResources, BufferSize);

// If the return value is anything other than NO_ERROR or
// ERROR_MORE_DATA, then exit the procedure.
if not((ReturnValue = NO_ERROR) or
   (ReturnValue = ERROR_MORE_DATA)) then begin
  case ReturnValue of
    NO_ERROR:
      ReturnValueStr := 'NO_ERROR';
    ERROR_NO_MORE_ITEMS:
      ReturnValueStr := 'ERROR_NO_MORE_ITEMS';
    ERROR_MORE_DATA:
      ReturnValueStr := 'ERROR_MORE_DATA';
    ERROR_INVALID_HANDLE:
      ReturnValueStr := 'ERROR_INVALID_HANDLE';
    ERROR_NO_NETWORK:
      ReturnValueStr := 'ERROR_NO_NETWORK';
    ERROR_EXTENDED_ERROR:
      ReturnValueStr := 'ERROR_EXTENDED_ERROR';
  end;

  MessageDlg(ReturnValueStr, mtError, [mbOk], 0);
  Exit;
end;

// Enter loop to continually enumerate the resource until
// all resources have been acquired.
repeat
  I := 0;
  // Obtain the resources that were enumerated with the
  // first call to WNetEnumResource.
  for I := 0 to ResourceEntries - 1 do begin
    NetResB := TNetResA.Create;
    NetResB.dwScope := ComputerResources^[I].dwScope;
    NetResB.dwType := ComputerResources^[I].dwType;
    NetResB.dwDisplayType :=
      ComputerResources^[I].dwDisplayType;
    NetResB.dwUsage := ComputerResources^[I].dwUsage;
    NetResB.lpLocalName :=
      StrPas(ComputerResources^[I].lpLocalName);
    NetResB.lpRemoteName :=
      StrPas(ComputerResources^[I].lpRemoteName);
    NetResB.lpComment :=
      StrPas(ComputerResources^[I].lpComment);
    NetResB.lpProvider :=
      StrPas(ComputerResources^[I].lpProvider);
    NetResB.NetResOwner := NetResOwner;
    NetResList.AddObject(IntToStr(I) + ' - ' +
      NetResB.lpRemoteName,NetResB);
  end;

  // Free the memory for the buffer and re-initialize
  // the variables.
  FreeMem(ComputerResources, BufferSize);
  ResourceEntries := MaxValue;
  BufferSize := 0;
  // Call WNetEnumResource again to determine the amount
  // of memory that is needed for the next enumeration of
  // network resource array information.
  ReturnValue := WNetEnumResource(NetResHand,
    ResourceEntries, ComputerResources, BufferSize);
  // Allocate the memory and re-initialize the
  // ResourceEntries variable.
  GetMem(ComputerResources, BufferSize);
  ResourceEntries := MaxValue;
  // Call WNetEnumResource again to continue the
  // enumeration of the network resource.
  ReturnValue := WNetEnumResource(NetResHand,
    ResourceEntries, ComputerResources, BufferSize);

until not((ReturnValue = NO_ERROR) or
         (ReturnValue = ERROR_MORE_DATA));

// Dispose of the pchar memory that was allocated for the
// PNetResourceA structure.
if Assigned(NetResOwner) then begin
  StrDispose(NetResA.lpLocalName);
  StrDispose(NetResA.lpRemoteName);
  StrDispose(NetResA.lpComment);
  StrDispose(NetResA.lpProvider);
end;

// Try to free the last memory allocated.
try
  FreeMem(ComputerResources, BufferSize);
except
end;

// Close the handle to that was opened with the call
// to WNetOpenEnem.
ReturnValue := WNetCloseEnum(NetResHand);
end;

end.
```

End Listing One

By Ron Loewy

Dynamic Control

Using the DHTML Edit Control from Delphi

When Microsoft shipped Windows 95, a new control for editing rich text was added to the multi-line edit control that shipped with Windows 3.x. The rich text control allowed the display and editing of formatted text using Microsoft's .RTF (rich text format) file format. .RTF was born when Microsoft needed a text-based format that would be interchangeable between the Macintosh and Windows versions of Microsoft Word. .RTF became important to many developers because Microsoft's Windows Help engine required .RTF files as the source of content that was compiled to .HLP files.

By the time Windows 95 arrived, however, the Web was gaining acceptance and HTML was becoming popular. Today, HTML is a standard method for exchanging rich text and hypermedia formats. To meet this standard, Microsoft presented the Dynamic HTML (DHTML) Control SDK — a WYSIWYG HTML editor that takes advantage of DHTML capabilities like absolute positioning of elements, access to the document object model, drag-and-drop capabilities, search, hyperlinks, image support, design-time controls, table support, and more.

The DHTML Edit Control Architecture

The DHTML Edit Control (DEC) uses the same HTML layout engine as Microsoft Internet Explorer (IE4). Because the same layout engine is used for the browser and the editing control, we're assured the editor will be very close to the actual output as it's displayed in the browser. IE4's HTML layout engine is defined in the file mshtml.dll, which implements an Active Document (DocObject) that renders HTML and handles ActiveX control embedding, Java VM hosting, plug-in hosting, and Active Script activation. DEC implements a new DocObject in the file, triedit.dll, that aggregates mshtml.dll and provides browsing and editing.

Internally, triedit.dll uses mshtml.dll for the rendering and layout of the code. Microsoft implemented an ActiveX control that encapsulates this DocObject and offers an easier way to use the visual editing control. The ActiveX control is implemented in the file, dhtmled.ocx.

Microsoft provides DEC free of charge. Your application will require that the user's machine have IE 4.01 or later, and you'll need to distribute the triedit.dll and dhtmled.ocx files that are part of the DEC SDK. You can download the DHTML Edit Control SDK from http://www.microsoft.com/workshop/author/dhtml/edit.

The DHTML Edit SDK

The DHTML Edit SDK documents how to use the DocObject or ActiveX control. The DocObject provides additional functionality and finer control of the editing environment compared to the ActiveX version of the control. Unfortunately, Delphi's implementation of *TOleContainer* in OleCtnrs.pas (in Delphi 3.02) doesn't work correctly with DocObjects and has to be heavily modified to perform as advertised. This article won't get into the changes you need to make to use the DocObject version of the control. Instead it will concentrate on using the ActiveX version, which is easier to use and provides almost all the capabilities offered by the DocObject control.

Installing DEC

You'll need to start by adding the control to Delphi's Component palette. Choose **Component | Import ActiveX Control**, and click on **DHTML Edit Control (Version 1.0)** to install it (see Figure 1). Choose the unit directory and palette you want to install the control to, and click the **Install** button.

If you followed the defaults during installation, two new components will be added to the ActiveX palette of Delphi. *TDHTMLEdit* is the ActiveX control we'll use. *TDHTMLSafe* is a safe version of the control that you can use in browser pages. The difference between the two controls is that the safe version doesn't provide access to the local file system, and is therefore safe for use over the Internet.

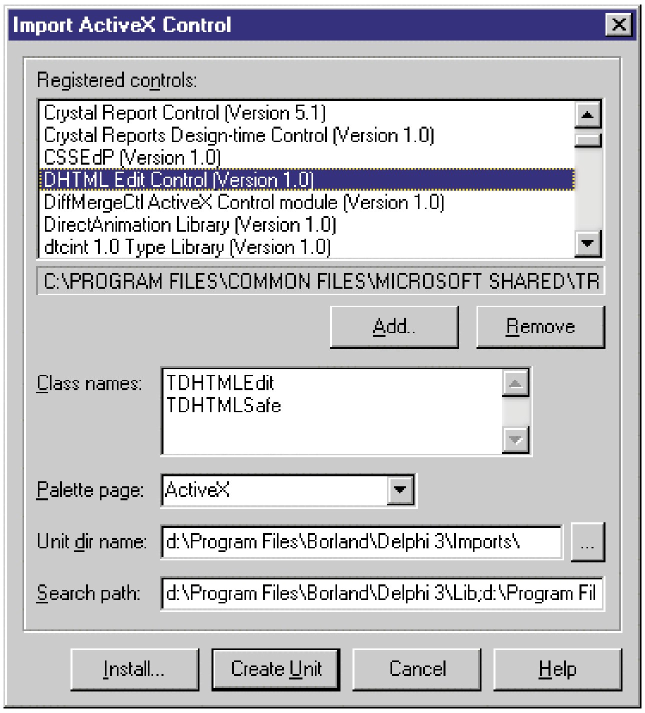
DEC provides a set of properties, events, and methods that allow you to perform tasks common to HTML editing. Following are some of these tasks and their related controls.

Document management. The *NewDocument* method allows you to start a new empty document for editing. The *LoadDocument* and *SaveDocument* methods can be used to load and save HTML documents to and from the file system. The *CurrentDocumentPath* property is used to determine the path to the document being edited. If you want to open documents that aren't stored on the file system, DEC supports the *LoadUrl* method that allows you to open a file from the Internet or intranet.

The *OnDocumentComplete* event is called when a new document has been created or loaded. When this event is called, you know that it's safe to access DEC to gain access to the DOM interface.

Editing content and edit tool notification. DEC provides a WYSIWYG editor interface where your users can edit the page contents. It's often useful to edit the control content programmatically. It allows access to the raw HTML code of the page (see the next section), or access to the DOM via the *ExecCommand* method (or the *IOleCommandTarget* interface).

Most applications offer users a visual way of issuing commands to DEC: a menu, a toolbar, and a floating toolbox are some of the more common. It's easy to connect your menu/toolbar button/... action to the *ExecCommand* method previously mentioned. However, it's important to ensure you're not issuing a bogus command. The *QueryStatus* method allows you to query the control to ensure that the command in which you're interested is enabled in the current control status. It makes no sense to send a "Bring to Front" command to an element that's not positioned absolutely. If you try to use *QueryStatus* before you call the *ExecCommand* with Bring to Front, you'll be notified if the command is valid for the edited element.


Import ActiveX Control

Registered controls:
Crystal Report Control (Version 5.1)
Crystal Reports Design-time Control (Version 1.0)
CSSEdP (Version 1.0)
DHTML Edit Control (Version 1.0)
DiffMergeCtl ActiveX Control module (Version 1.0)
DirectAnimation Library (Version 1.0)
dtcint 1.0 Type Library (Version 1.0)
C:\PROGRAM FILES\COMMON FILES\MICROSOFT SHARED\TR

Class names: TDHTMLEdit
TDHTMLSafe
Palette page: ActiveX
Unit dir name: d:\Program Files\Borland\Delphi 3\Imports\
Search path: d:\Program Files\Borland\Delphi 3\Lib;d:\Program Fil

Figure 1: Importing the DHTML Edit Control in Delphi.

The *OnDisplayChanged* event is called whenever a new element is written, chosen, or selected in the control. Your code should handle this event, and use *QueryStatus* and other methods to update your visual tool displays. For example, if the selected text is in bold, you would like to display your bold button in the "down" position.

HTML source access. Although DEC allows your users to work in WYSIWYG mode and avoid the details related to HTML code, your application will sometimes be better off when you work with the raw code. For example, if you allow the user to edit multiple HTML pages from different pages, it's important to remember to set the *BaseUrl* property of the control to the directory the page was loaded from, or the control won't display images correctly for IMG tags that use relative paths to the image source.

The *DocumentHTML* property allows you to access the HTML source code of the page as a long string. You can also set the page contents by assigning a valid HTML page code to this property. It's a good idea to check the *Busy* property before accessing *DocumentHTML*. The control might be in an unstable condition when editing, parsing, or loading occurs. At these stages, the *DocumentHTML* property might not be valid.

Selections. Editing in DEC is done using selection objects. Selection objects are obtained via the Document Object Model the control exposes using the *DOM* property. The DOM's selection object can be used to inspect the selected code by creating a *TextRange* object. (Use `TextRange := DHTMLEdit1.DOM.selection.CreateRange` to obtain the *TextRange* object. This object's *HtmlText* property provides the source code of the selected area. A complete description of the selection and text range objects is available in the Internet Client SDK.) The selection object's *Type* property can tell you if the selected object is a regular HTML stream, or a control embedded in the page.

Control entry points. DEC's *DOM* property allows you to access the DOM hierarchy for the control. It serves the same purpose as the *Document* property of *TWebBrowser*. (See my article "IE4's DOM Advantage" in the August, 1998 issue of *Delphi Informant*.) You can be sure the *DOM* property points to a correct *IHtmlDocument2* interface implementation when the control fires the *OnDocumentComplete* event.

I declare the following variables in the form that hosts my DEC:

```
DocComplete   : Boolean;
CommandTarget : IOleCommandTarget;
DOMInterface  : IHtmlDocument2;
```

In the *OnDocumentComplete*, I obtain the following pointers:

```
procedure TDHEDForm.DHTMLEdit1DocumentComplete(
  Sender: TObject);
begin
  DocComplete := True;
  DOMInterface := DHTMLEdit1.DOM as IHtmlDocument2;
  CommandTarget := DOMInterface as IOleCommandTarget;
end;
```

CommandTarget provides a way to send commands to the control via the standard *IOleCommandTarget* interface.

Creating a WYSIWYG editor application. To demonstrate DEC's use from a Delphi application, and to discuss some of the interesting techniques of working with the control, we'll develop a simple HTML editor that uses the control (available for download; see end of article for details). Our application will be a simple SDI application with a menu bar, a toolbar, the editing area taken by the edit control, and a status bar.

```
procedure TDHEDForm.OpenBtnClick(Sender: TObject);
var
 pVIn : OleVariant;
 Prompt : OleVariant;
begin
  pVIn := '';
  Prompt := True;
  DocComplete := False;
  DHtmlEdit1.LoadDocument(pVIn, Prompt);
end;
```

Figure 2: The code behind the **Open** button.

```
procedure TDHEDForm.SaveBtnClick(Sender: TObject);
var
  vo, vb : OleVariant;
begin
  vo := DHTMLEdit1.CurrentDocumentPath;
  if (vo <> '') then
     vb := False
  else
    begin
      vo := '';
      vb := True;
    end;
  DHTMLEdit1.SaveDocument(vo, vb);
end;
```

Figure 3: Code used to implement the *Save* function.

```
BoldBtn.Down      := ((QueryStatus(DECMD_BOLD) and
                       DECMDF_LATCHED) = DECMDF_LATCHED);
ItalicBtn.Down    := ((QueryStatus(DECMD_ITALIC) and
                       DECMDF_LATCHED) = DECMDF_LATCHED);
UnderlineBtn.Down := ((QueryStatus(DECMD_UNDERLINE) and
                       DECMDF_LATCHED) = DECMDF_LATCHED);
CutBtn.Enabled    := ((QueryStatus(DECMD_CUT) and
                       DECMDF_ENABLED) = DECMDF_ENABLED);
CopyBtn.Enabled   := ((QueryStatus(DECMD_COPY) and
                       DECMDF_ENABLED) = DECMDF_ENABLED);
PasteBtn.Enabled  := ((QueryStatus(DECMD_PASTE) and
                       DECMDF_ENABLED) = DECMDF_ENABLED);
```

Figure 4: Code used to invoke the *QueryStatus* method.

```
FontNameStatus := GetCommandStatus(IDM_FONTNAME, False);
FontSizeStatus := GetCommandStatus(IDM_FONTSIZE, False);
if ((FontNameStatus and OleCmdf_Enabled) <> 0) then
  begin
    HrExecCommand(IDM_FONTNAME, NilVariant,
                  vo, False, False);
    if (VarType(vo) = VarOleStr) then
      TEFontNameBox.ItemIndex :=
        TEFontNameBox.Items.IndexOf(Vo);
  end;
  VarClear(vo);
  if ((FontSizeStatus and OleCmdf_Enabled) <> 0) then
    begin
      HrExecCommand(IDM_FONTSIZE, NilVariant,
                    vo, False, False);
      if (VarType(vo) = VarInteger) then
        TEFontSizeBox.ItemIndex :=
          TEFontSizeBox.Items.IndexOf(vo);
    end;
```

Figure 5: Code used to invoke font name and size boxes.

The toolbar will include file management buttons for **File | New**, **File | Open**, and **File | Save**, as well as the **Cut**, **Copy**, and **Paste** edit buttons, and **Undo** and **Redo**. **Find** and text-formatting buttons that show the selected text status and can issue commands are also included. (For our sample we'll include bold, italic, and underline buttons, font name and font size drop-down boxes, and an HTML styles box.)

File management. We'll connect the **New** button (and the **File | New** menu command) to DEC's *NewDocument* method. The **Open** button is connected to the control's *LoadDocument* method, and the **Save** button is connected to the *SaveDocument* method. Figure 2 shows the **Open** button code.

Parameters are passed to control methods as Variants. We must initialize these parameters before we use them. Notice that we set the *DocComplete* object to False. It will be set to True when the control finishes loading the document and handles the *OnDocumentComplete* event we just discussed. The *Save* function is implemented using the code shown in Figure 3.

Toolbar and menu format updates. In an edit application, toolbar controls can usually be in two states: initialization and active. When the controls are in the initialization state, the tools shouldn't apply any changes to the edit control when their values are set. After defining the global form variable *TEIgnoreChange* and setting it to True, the toolbar controls are in initialization state. We initialize the font names box using the following code in the *FormShow* procedure:

```
procedure TDHEDForm.FormShow(Sender: TObject);
begin
  TEIgnoreChange := True;
  TEFontNameBox.Items.Assign(Screen.Fonts);
  TEIgnoreChange := False;
end;
```

We now need to update the toolbar controls whenever the user moves the cursor in the application. We need to trap the edit control's *OnDisplayChange* event and update our toolbar display. The cut, copy, paste, bold, italic, and underline buttons are simple. We use the *QueryStatus* method using the code shown in Figure 4. The font name and size boxes are a bit more complicated (see Figure 5).

GetCommandStatus activates *QueryStatus* via the *IOleCommandTarget* interface. (I just wanted to demonstrate that you can access the *QueryStatus* method using a different approach.) We later use the *IOleCommandTarget*'s *Exec* method in *HrExecCommand* to get the font name and size and update the listboxes with them. Notice that when we need to pass a **nil** value to the *Exec* command, we use the variable *NilVariant* that is defined using the following syntax:

```
var
  NilVariant : OleVariant absolute 0;
```

When you need to send a null (**nil**) value to *IOleCommandTarget*, *IOleCommandTarget* expects it to point to a value of 0. The reserved word **absolute** does the trick in Delphi.

Updating the **HTML Style** box is done the same way, with the exception that if the box has 0 items in it (the first time we update the selection format settings), we need to fill the box. This is done with the *GetAvailableStyles* method, which is worth checking out because it demonstrates how Delphi applications can access safe arrays (see Figure 6).

```
procedure TDHEDForm.GetAvailableStyles;
var
  varRange : OleVariant;
  b        : TBStr;
  a        : PSafeArray;
  l, h, i  : Longint;
  hr       : HRESULT;
begin
  TVariantArg(VarRange).VT := VT_ARRAY;
  TVariantArg(VarRange).ppArray := nil;
  hr := HrExecCommand(IDM_GETBLOCKFMTS, NilVariant,
                      VarRange, False, False);
  if (hr = 0) then
    begin
      l := VarArrayLowBound(VarRange, 1);
      h := VarArrayHighBound(VarRange, 1);
      a := TVariantArg(VarRange).pArray;
      for i := l to h do begin
        SafeArrayGetElement(a, i, b);
        TEStylesBox.Items.Add(OleStrToString(b));
      end;
    end;
end;
```

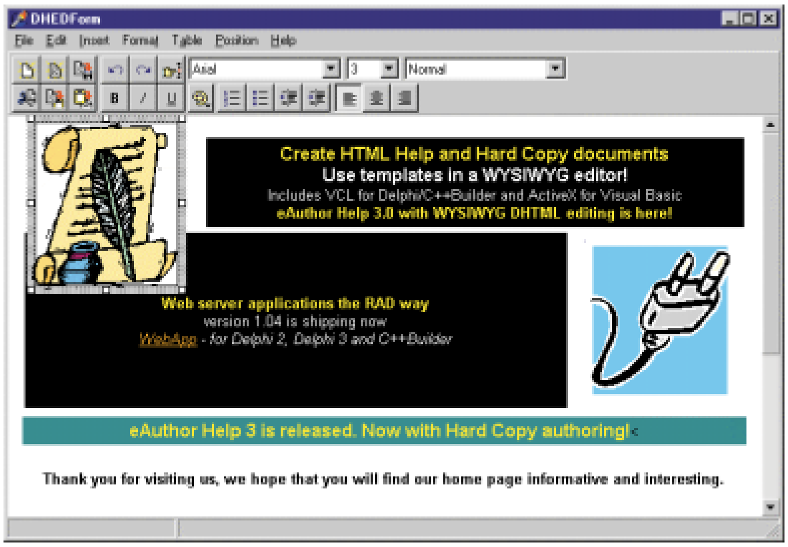
Figure 6: The *GetAvailableStyles* method demonstrates how Delphi applications can access safe arrays.

A safe array is stored in an OleVariant; unfortunately you need to go through a couple of hoops to gain access to safe array elements in Delphi. Because many Visual Basic and other OLE/COM technologies require parameters (or return results) in safe arrays, it's useful to keep this sample around.

The *OnDisplayChange* event uses the same techniques to update all the other toolbar buttons and menu commands.

Simple editing commands. We'll use the DEC's *ExecCommand* method to activate simple commands like cut, copy, and paste. I wrapped the following call with my own *ExecCommand* procedure, thus ensuring the control isn't busy, and the command I want to execute is valid:

```
procedure TDHEDForm.ExecCommand;
begin
  if (DocComplete) then
    if ((QueryStatus(CmdID) and DECMDF_ENABLED) =
        DECMDF_ENABLED) then
      DHTMLEdit1.ExecCommand(cmdID, cmdExecOpt, pVar);
end;
```



Create HTML Help and Hard Copy documents
Use templates in a WYSIWYG editor!
Includes VCL for Delphi/C++Builder and ActiveX for Visual Basic
eAuthor Help 3.0 with WYSIWYG DHTML editing is here!

Web server applications the RAD way
version 1.04 is shipping now
WebApp - for Delphi 2, Delphi 3 and C++Builder

eAuthor Help 3 is released. Now with Hard Copy authoring!

Thank you for visiting us, we hope that you will find our home page informative and interesting.

Figure 7: An image element is positioned absolutely in the sample application.

Using *ExecCommand* is now simple. For example, the following code will perform the cut operation:

```
ExecCommand(DECMD_CUT, OLECMDEXECOPT_DODEFAULT, Null);
```

The undo, redo, and bold commands use the *TridentCommand* procedure that calls *HrExecCommand* (same as the control's *ExecCommand* method, only using the *IOleCommandTarget* interface). The *TridentCommand* method allows you more control over the command parameters. We'll see a sample of this later when we discuss table handling.

All other commands exposed under the **Edit**, **Insert**, and **Format** menus use the same techniques to send a command to the control.

Table commands. Adding table support to a visual editor is a task most developers would prefer to avoid. If writing a visual WYSIWYG editor is hard, adding table support complicates it ten-fold. Fortunately, the DHTML Edit control provides support for table editing.

The **Table** menu in our application uses the *TridentCommand* method to create tables and insert or delete rows, columns, and cells, as well as perform operations like cell merging and splitting. For the table creation function, we take advantage of the *GetInputArg* parameter of the *TridentCommand* method and call an external method that displays the table attributes dialog. The table parameters collected in the dialog are packed into a *Variant* variable, and passed to the *IOleCommandTarget's ExecCommand* method via the *HrExecCommand* wrapper in our form source.

Absolute positioning. Traditional HTML is a stream-based layout language, such as .RTF and most word processors we have used. Page layout programs use an absolute positioning of elements (see Figure 7).

Trying to combine stream-based content with absolute positioning of elements in an editing application is hard to do. Word 97 introduced a "graphic" layer on top of the standard content stream (if you ever tried to position images in Word 97 documents accurately, you'll understand how complicated this can be). However, Word has to deal with printed output where the positioning of graphic elements in relation to the text stream can vary with different printers.

HTML, on the other hand, was designed for online document delivery, and DHTML provides the ability to include absolute positioning of elements with pixel precision using Cascading Style Sheets. DEC supports this function, and we'll include it in our sample application.

DEC can't position every element on the page absolutely. Only the following HTML elements can be positioned absolutely: <APPLET>, <BUTTON>, <EMBED>, <DIV>, <EMBED>, <HR>, <IFRAME>, <IMG>, <INPUT>, <MARQUEE>, <OBJECT>, <SELECT>, <SPAN>, <TABLE>, <TEXTAREA>, and <FIELDSET>. Other elements are always part of the HTML stream.

We determine if an element can be positioned absolutely using the *GetTridentCommandState* method with the IDM_TRIED_MAKE_ABSOLUTE control command. See the event for the **Position** menu that checks for this information.

```
procedure TDHEDForm.TranslateAcceleratorHandler;
const
  DuplicatedKeys: set of Byte =
    [VK_BACK, VK_LEFT, VK_RIGHT, VK_UP, VK_DOWN,
     VK_PRIOR, VK_NEXT];
var
  iOIPAO: IOleInPlaceActiveObject;
  Dispatch: IDispatch;
begin
  if (Assigned(DHTMLEdit1)) then
    begin
      Handled :=
        (IsDialogMessage(DHTMLEdit1.Handle, Msg) = True);
      if ((Handled) and (not DHTMLEdit1.Busy)) then
        begin
          if (FOleInPlaceActiveObject = nil) then
            begin
              Dispatch := DHTMLEdit1.ControlInterface;
              if (Assigned(Dispatch)) then
                begin
                  Dispatch.QueryInterface(
                    IOleInPlaceActiveObject, iOIPAO);
                  if (Assigned(iOIPAO)) then
                    begin
                      FOleInPlaceActiveObject := iOIPAO;
                      iOIPAO._Release;
                    end; // Have an active inplace object.
                end; // Have dispatch.
            end; // Need active inplace object.
          if (Assigned(FOleInPlaceActiveObject)) then
            if ((Msg.message = WM_KEYDOWN) or
                (Msg.message = WM_KEYUP)) and
                (Msg.wParam in DuplicatedKeys) then
              // Do nothing; don't pass on cursor movement
              // keys or they will happen twice.
            else
              FOleInPlaceActiveObject.
                TranslateAccelerator(Msg);
        end;
    end
  else
    Handled := False;
end;
```

Figure 8: The message handler resulting from overcoming shortcut key problems.

Absolutely positioned elements have both 2D coordinates (*x*, *y*) and a *z*-order coordinate. The latter is used to determine which element will be displayed if more than one element occupies the same area. Elements that have a closer *z*-order will have priority and be displayed over elements that are farther away.

The Position menu includes a set of menu items that handle the *z*-order position of an element, including Send to back, Bring to front, Send backward, Bring forward, Send behind HTML stream, and Bring above HTML stream.

The Nudge element option allows you to move an object a specified number of pixels (you'll usually just drag and drop the element with the mouse, but sometimes you need to ensure alignment, etc.), and the Lock element option allows an object to be locked in place and not be moved by mistake. When the Constrain element positioning option is checked, you can move an element in one dimension only. If you start moving the object up, you will not lose your *x*-axis placement by mistake.

Fixing Delphi Glitches

DEC is implemented as an ActiveX control that hosts a DocObject. This isn't a common way to create controls, and Delphi's implementation of control containers (the form code) doesn't work with it too well. The clash between Delphi's implementation of the embedding code and the control results in problems with shortcut key activation from within the control and some functions (most notably the cut, copy, and paste functions) of the control not working correctly.

Fixes to these problems were researched by several developers with the help of the Microsoft DEC development team, and Henri Fournier deserves most of the credit to the solutions. To ensure that the cut, copy, and paste functions work, you need to add the following **initialization** and **finalization** sections to the unit of the form that hosts the control:

```
initialization
  OleInitialize(nil);

finalization
  OleUninitialize;
```

To overcome the shortcut key problems in the form's *OnActivate* event, you'll need to add the following code to the form's unit:

```
procedure TDHEDForm.FormActivate(Sender: TObject);
begin
  OldMessageHandler := Application.OnMessage;
  Application.OnMessage := TranslateAcceleratorHandler;
end;
```

and, on deactivation, restore the message handler:

```
procedure TDHEDForm.FormDeactivate(Sender: TObject);
begin
  Application.OnMessage := OldMessageHandler;
end;
```

The new message handler is shown in Figure 8.

Remember, Microsoft provides DEC free of charge. Your application will require that the user's machine have IE 4.01 or later, and you'll need to distribute the triedit.dll and dhtmled.ocx files that are part of the DEC SDK (which you can download from http://www.microsoft.com/workshop/author/dhtml/edit).

Conclusion

With the DHTML Edit Control, every application can provide visual HTML editing capabilities. Delphi is a great choice for applications that need to use ActiveX controls. And if you're willing to get your fingers greasy by using OLE and COM interfaces, you can take advantage of the powerful capabilities Microsoft provided with the DHTML Edit Control. ∆

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\99\AUG\DI9908RL.

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 Motty Adler

Rectangles

A Closer Look at the *TRect* Windows Data Type

Rectangles are a major part of Windows life. Indeed, all visible VCL components have *Top*, *Left*, *Right*, and *Height* properties that form a rectangle. All window objects in Windows are described as rectangles. Thus, the Windows API has a large group of functions that use and manipulate rectangles. The Delphi VCL adds many new ways to use rectangles.

The basic RECT is described by Windows as a structure of four integers: *top*, *left*, *bottom*, and *right*. The following is the C declaration found in the API Help:

```
typedef struct _RECT
{
  LONG left;
  LONG top;
  LONG right;
  LONG bottom;
} RECT;
```

In Delphi, this structure is defined as a **record** of type *TRect*. This is its declaration:

```
TRect = record
  case Integer of
    // This is the standard part of the
    // record.
    0: (Left, Top, Right, Bottom: Integer);
    // We'll talk about this later.
    1: (TopLeft, BottomRight: TPoint);
end;
```

This article attempts to provide a comprehensive look at this important Windows data type. It will detail how and when it's used. It will also take a short spin through some API functions — some not widely known — to show how you can fully utilize *TRect*.

In its default form, *TRect* contains the same four integers: left, top, right, and bottom. It's generally similar to RECT, and can be used in one way or another whenever the API calls for a RECT.

A *TRect* defines an area bounded by left, top and right, bottom. This area does not have to refer to actual window real estate, but can represent a virtual rectangle as well. The values in the rectangle are offsets of the top, left. The top, left of the coordinate system is 0,0, and that increases as the distance moves down or right. The size of the rectangle is the difference between the furthest edge and the closest edge. For example, if the rectangle is X=25,Y=25 to X=50,Y=50, then the rectangle is 25 high by 25 wide.

Basic *TRect* Manipulation

Because rectangles are so important, Windows has many functions to manipulate and analyze them. To use a *TRect*, you declare it in the **var** section of a unit or procedure. As with all Delphi variables, the members of the *TRect* will already have random values assigned to them. If you want to set all the values to zero, you can set all the members manually, or you can use an API function named *SetRectEmpty*, that "empties" the rectangle. A *TRect* is considered empty when there can be no space in it. If the bottom value is less or equal to the top value, or the right is less or equal to the left, the rectangle is considered empty.

SetRectEmpty takes the *TRect* as a parameter and changes all its member values to zero. You would call *SetRectEmpty* like this:

```
var
  Rec: TRect;
...
begin
  SetRectEmpty(Rec);
end;
```

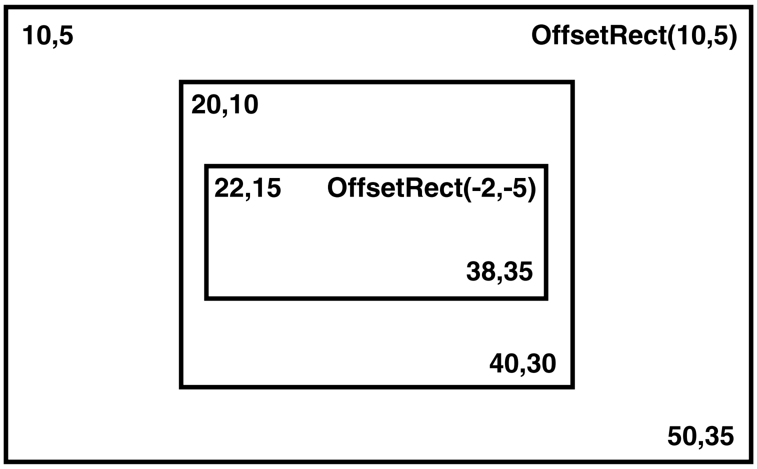


Figure 1: *InflateRect* inflates the rectangle from all sides.

Most API functions will not handle an empty rectangle and will just ignore it. To determine if a *TRect* is empty, you can use the API's *IsRectEmpty* function. *IsRectEmpty* takes the *TRect* as a parameter and returns a Boolean True if the *TRect* is empty, and a False if it isn't.

You can fill the *TRect*'s members by assigning them directly. For example:

```
Rec.Left := 30;
```

However, the API provides a quicker way to fill the *TRect*'s members. It's called *SetRect*, and takes as parameters the *TRect* and four signed integers, for top, left, bottom, and right. This frees the programmer from having to manually fill all the members. Delphi has an even better way to do this: the *Rect* function. The *Rect* function also takes the four integer parameters, but instead of modifying an existing *TRect*, it returns a *TRect* as its result. Thus, to assign a value to a *TRect*, you can use code like this:

```
Rec := Rect(30,23,173,142);
```

Because *Rect* returns a 16-byte array, it has an advantage over the API function. In the API approach, if you need to use a *TRect* as a parameter to a function, you have to declare it, assign the value using *SetRect*, then pass the *TRect* to the function:

```
function DoSomeTRectThing;
var
  Rec:TRect;  // We have to define a TRect;
begin
  SetRect(Rec, 30, 23, 173, 142);
  ProcessTRect(Rec);
end;
```

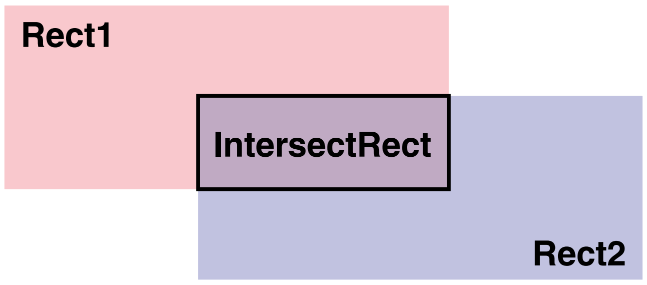


Figure 2: Overlapping rectangles share a rectangular area, and *IntersectRect* fills a *TRect* with the coordinates of this rectangle.

The Delphi approach allows you to use the *Rect* function as a parameter to a procedure:

```
function DoSomeTRectThing;
begin
  ProcessTRect(Rect(0, 23, 173, 142));
end;
```

Note, however, that this will not work with a function that wants to modify the value of the *TRect*, or with a function that requires a *PRect* (we'll discuss this in more detail later).

Copying the values of one *TRect* to another can be done using the *CopyRect* API function. *CopyRect* takes the source and the destination rectangles as parameters. However, because *TRect* is a standard Delphi type, all you have to do is assign one *TRect* to another, using the assignment operator (:=).

When it comes to comparing *TRect* objects, Delphi won't allow a direct comparison, i.e. with the = operator. So instead of making a long-winded "if ladder" (`if top = top then if left =` ... etc.), you could use the API *EqualRect* function, which takes the two *Rect* objects and returns a Boolean. A call to *EqualRect* looks like this:

```
if EqualRect(Rect1, Rect2) then
  DoSomething  // They are equal.
else
  DoSomeThingElse;
```

To change the width and length of a *TRect*, you can use the API *InflateRect* function. You pass the *TRect* and the *x* and *y* amount of pixels you want changed. If the value is positive, the rectangle is enlarged; if it's negative, the rectangle is made smaller by that amount.

InflateRect inflates the rectangle from all sides. If the *TRect* you passed is 10,10 - 20,20, and you passed 5 for *x* and 10 for *y*, the resulting rectangle will be 5, 0, 25, 30 (see Figure 1).

Moving a whole *TRect* can be done by using the API *OffsetRect* function; pass an *x* and a *y* to move the *TRect* by that amount. *OffsetRect* adds that amount to each member value so that if you pass a positive, the rectangle moves to the right and down; if you pass a negative, it moves left and up.

Advanced *TRect* Manipulation

The Windows API provides functions for more advanced manipulation of rectangles. The *PtInRect* function will test if a certain position is in a rectangle. *PtInRect* takes a *TRect* and a *TPoint* (also a Delphi **record** type, similar to a *TRect*, and describes only one point as *x* and *y*). If the *TPoint* is in the *TRect*, then the *PtInRect* returns True; otherwise, it returns False.

Rectangles don't have to be alone in a coordinate system. You can have many rectangles floating around in the same virtual space. Some rectangles may overlap and intersect with others. The Windows API provides functions that analyze the positions of two *TRect* objects in relation to each other, and returns a rectangle that holds the result.

The following API functions take three parameters. The first is the rectangle to fill with the result; the next two are the rectangles to be analyzed. Overlapping rectangles share a rectangular area, and *IntersectRect* fills a *TRect* with the coordinates of this rectangle (see Figure 2).

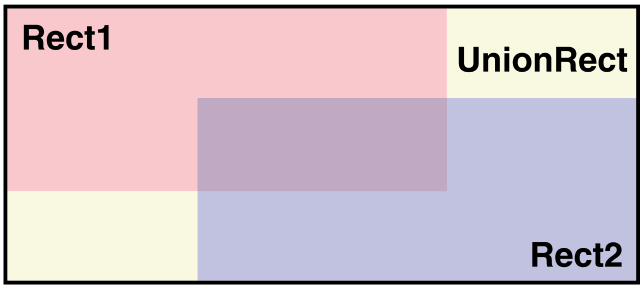


Figure 3: You can use the *UnionRect* function to retrieve a rectangle that fully encompasses both rectangles — even if they don't overlap.

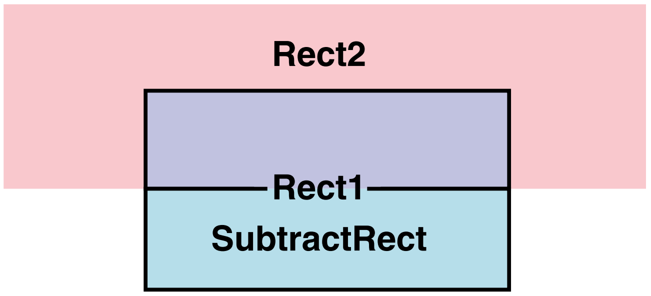


Figure 4: The *SubtractRect* function will return the part of the second rectangle that is not covered by the first. This can only be a rectangular area if the second rectangle totally covers the first in either width or height.

Function	Use
CopyRect	Copies the values of one *TRect* to another.
InflateRect	Enlarges the rectangle from all sides.
IntersectRect	Returns the overlapping area of two rectangles.
IsRectEmpty	Determines if a rectangle does not contain space.
OffsetRect	Moves the entire rectangle.
Rect	VCL — Sets all the member values at once.
SetRect	API — Sets all the member values at once.
SetRectEmpty	Sets all the rectangles members to 0.
SubtractRect	Returns the area of the first rectangle that is not covered by the second.
UnionRect	Returns the smallest rectangle that can hold both rectangles.

Figure 5: *TRect* manipulation functions.

```
function ScreenToClientRect(Rec: TRect;
  Control: TControl): TRect;
begin
  Result.TopLeft := Control.ScreenToClient(Rec.TopLeft);
  Result.BottomRight :=
    Control.ScreenToClient(Rec.BottomRight);
end;

function ClientToScreenRect(Rec: TRect;
  Control: TControl); TRect;
begin
  Result.TopLeft := Control.ClientToScreen(Rec.TopLeft);
  Result.BottomRight :=
    Control.ClientToScreen(Rec.BottomRight);
end;
```

Figure 6: Converting screen and client coordinates.

You can use the *UnionRect* function to retrieve a rectangle that fully encompasses both rectangles — even if they don't overlap (see Figure 3).

The *SubtractRect* function will return the part of the second rectangle not covered by the first. This can only be a rectangular area if the second rectangle totally covers the first in either width or height, as illustrated in Figure 4. If this is not True, then *SubtractRect* ignores the actual subtraction and simply returns the coordinates of the first *TRect*. This is something you have to watch out for because it can mess up your program's logic. It's recommended you test the results using *EqualRect* (unless that's the result you want).

All these functions are demonstrated in the included sample application (see the end of this article for download details). Figure 5 is a table with all the rectangle manipulation functions and their tasks.

Extra *TRect* Goodies

A *TRect* is represented in memory as a block of data, 16 bytes long. This is divided into four integers of 4 bytes each. The Delphi *TPoint* is a block of memory 8 bytes long and divided into two integers describing an *x*, *y* position. Two *TPoint*s can fit into one *TRect*.

Because a *TRect* specifies two points, the left, top and the right, bottom, it's useful that Delphi allows variant parts in a record. Variant parts allow you to access parts of a record in different ways. Here are the variant parts, and what they equal:

Rect.Topleft.x = Rect.Left
Rect.TopLeft.y = Rect.Top
Rect.BottomRight.x = Rect.Right
Rect.BottomRight.y = Rect.Bottom

This is helpful in a case where you want a point of the *TRect* to get its value from a function that returns a *TPoint*. For example, *GetCursorPos* is an API function that fills a *TPoint* with the position of the mouse cursor in screen coordinates. To use the mouse position as the bottom-right part of a *TRect*, you could use this code:

```
GetCursorPos(MyRect.BottomRight);
```

Converting Screen Coordinates to Client Coordinates

API functions such as *GetCursorPos* return coordinates in offsets from the top left of the screen. To convert a *TRect* from screen coordinates to the client coordinates of a window, we can take advantage of the *TRect*'s variant parts (*TopLeft* and *BottomRight)*, and the *TControl.ClientToScreen* method. The listing in Figure 6 shows functions that convert screen coordinates to client coordinates, and vice versa.

What Is a *PRect*?

When you pass a *TRect* to a function as a parameter, you are passing a full 16 bytes containing the *TRect* to the function. This is only a copy of the original value. The function can inspect the values and do some action based on it, but it can't change the original *TRect*. This is what the *Rect* function does. It returns a 16-byte block of memory containing the values.

However, the calling function cannot change the values (as many of the API functions want to do). For a function to change the original *TRect*, you must pass it as the memory address of the record. The function can then look at the original *TRect* and change its values.

Delphi defines some of the API functions as expecting **var** *TRect*. Internally, whenever the compiler sees a function parameter declared as a **var**, it does not pass the actual record, but passes a memory address to the original record. This is why you can use a standard *TRect* to call API functions; all of which want the address of the *TRect*. However, some API definitions expect a variable type named *PRect*, which is a pointer to a *TRect*. Delphi 3 or 4 can show you which type is required with its Code Parameters tool hint window. To pass a *TRect* to a function requesting a *PRect*, you can use the @ (at) operator before the *TRect*'s name. The @ operator causes the variable to which it is prepended to act as a pointer, and have it passed as an address.

So what is the difference between **var** *TRect* and *PRect*, and why did Inprise define some parameters as **var** *TRect* objects and others as *PRect* objects? Well, according to Peter Below of TeamB, it seems that whenever an API function might require a **nil** to be passed to it, the function was defined as a *PRect* because a **var** *TRect* cannot accept a **nil**.

TRect in Everyday Life

So why is *TRect* so important? *TRect* describes a rectangular area, and many objects in Windows can be described as rectangles. Let's look at some Delphi/API functions that accept *TRect* objects as a parameter.

A window is a rectangle, and so are all VCL controls. A *TRect* can adequately describe them. Delphi provides the *BoundsRect* method for the *TControl* and its descendants. *BoundsRect* returns a *TRect* describing the position of the control within its parent window. To get the rectangular area of non-VCL Windows, you can use the API *GetWindowRect*, which fills a *TRect* with the description.

To set the size and position of a VCL control, you can change the *BoundsRect* property, or, for a non-VCL window, you can use the API *SetWindowRect* function.

The client rectangle of a window is the area not covered by the Windows border, the menu bar, or any other of the standard interface objects. To retrieve this rectangle, you can use the *ClientRect* property for VCL controls, or the *GetWindowRect* API function for others.

Clipping the cursor means that you force the cursor to stay within a certain part of the screen. This is a rectangle that is passed to the *ClipCursor* API function. The coordinates for the *TRect* are in offsets from the top, left of the screen. For any mouse movement, Windows ensures the mouse pointer does not move out of the rectangle. Because there is only one shared mouse for the whole system, the clipping rectangle affects all applications. To cancel the clipping rectangle, call *ClipCursor* again, this time passing it a **nil**. Because *ClipCursor* does at times take a **nil**, it's defined as a *PRect*, so to pass a *TRect* to *ClipCursor*, you'll have to use the @ operator.

It's important to make sure that the rectangle you pass to the *ClipCursor* is not empty. If it's empty, the cursor will be stuck in its original position and will remain stuck there until it is freed. Therefore, it's recommended that you test for *IsRectEmpty* before calling *ClipCursor*.

MDI (Multiple Document Interface) windows have a mechanism to cascade all open child windows. There isn't an easy way to cascade all the top-level windows. You can, however, use the function that MDI windows use internally to cascade its client windows: the *CascadeWindow* function. It takes a handle to the parent window. But if the handle is **nil**, the desktop window is assumed. You can pass an open-ended array of window handles to the function, and the function will only cascade those windows. The interesting attribute of this function for our discussion is that you can specify the rectangle in which the windows will be cascaded to.

TCanvas

Delphi provides the *TCanvas* property for many of its components. *TCanvas* is an encapsulation of a Windows device context. Device contexts are objects in Windows that represent the surface of a window or a bitmap. You draw to a window by drawing to its device context. The device context is essentially a structure (**record**) with information about the drawing state of the window. Pen, brush, and font information are stored in the device context. All windows use device contexts to draw themselves, including controls such as list boxes and buttons. Windows provides a number of functions to draw text and other shapes to a device context. Delphi encompasses many of these functions into properties and methods of the *TCanvas* object. Some components (such as the *TForm* and *TImage)* automatically surface the *TCanvas* property.

Whenever it's time for a window to redraw itself, Windows sends a message. Delphi converts this message to the *OnPaint* event and the control repaints itself. The graphical drawing routines can take a long time to execute, and there might be no need to repaint the whole control simply because a small corner got uncovered. Indeed, the whole window doesn't get repainted. Windows defines something called a Clipping Rectangle. This is the rectangle to which paint requests will be drawn. Any drawing out of this rectangle will be ignored.

On a *TCanvas*, this rectangle can be retrieved using the *ClipRect* property. You can also set this property. To determine if a certain rectangle is part of the clipping region, you can use the *RectVisible* function. The *TCanvas* encapsulates some of the Windows GDI (Graphics Device Interface) functions that make them easier to use. I will run through some that are somewhat related to *TRect*.

You can copy a whole image onto a *TCanvas* by using the *Draw* method. However, *Draw* copies the original image in its original height and width. You can use the *StretchDraw* method to specify the destination rectangle. The drawn graphics will be enlarged or shrunk to fit into the *TRect* you provided. Even better is the *CopyRect* method, which allows you to specify the source rectangle as well.

The *TextRect* function is like the *TextOut* function in that it draws text on to the *TCanvas* in the specified *x* and *y*. But, *TextRect* also allows you to specify a clipping rectangle to draw the text in. Any text that would output outside the bounds of this rectangle will not be drawn.

It's easier to handle a rectangle as a *TRect* than as separate *x*, *y*, *x*, *y* values. Most of the shape-drawing functions in Delphi — even the *Rectangle* method — take separate *x*, *y*, *x*, *y* variables. You can

```
procedure Draw3DBevelFrame(rct:TRect);
begin
  with canvas do begin
    brush.color := clBtnShadow;
    FrameRect(rct);
    brush.color := clBtnHighlight;
    OffsetRect(rct,-1,-1);
    FrameRect(rct);
  end;
end;
```

Figure 7: Drawing a bevel using *FrameRect*.

still use the *TRect* to draw a rectangle by using the *FrameRect* method. *FrameRect* draws a rectangle with one-pixel walls using the brush. To fill a rectangular area using the brush, you can call the *FillRect* method. The code listing in Figure 7 shows a quick way to draw a three-dimensional bevel frame using *FrameRect*.

When a control has the focus, it's expected to have some kind of visual indicator showing this. Most controls draw a dotted rectangle around them. The API has a function *DrawFocusRect* that does this, and Delphi encapsulates this as the *DrawFocusRect* method. This method draws the dotted line in an XOR pen; every second pixel is inverted. This allows you to erase the line by redrawing it again. The listing in Figure 8 shows the procedures you can use to add a rubber-band-style box to your application.

```
// Form-level variables.
var
  rec: TRect;

procedure TForm2.FormMouseDown(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  rct.TopLeft.x := x;  // These will be the origin
  rct.TopLeft.y := y;  // of the rectangle.
  // Erase any BottomRights left from last time.
  rct.BottomRight := rct.TopLeft;
end;

procedure TForm2.FormMouseMove(Sender: TObject;
  Shift: TShiftState; X, Y: Integer);
begin
  if not (ssLeft in Shift) then
    Exit;  // Mouse is not down.
  // Erase the last rectangle by drawing over it.
  Canvas.DrawFocusRect(rct);
  // Set the new bottom-right position.
  rct.Bottom := y;
  rct.Right := x;
  Canvas.DrawFocusRect(rct);  // Draw the new rectangle.
end;

procedure TForm2.FormMouseUp(Sender: TObject;
  Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
  Canvas.DrawFocusRect(rct);  // Erase for last time.
end;
```

Figure 8: Implementing a rubber-band box.

Drawing with the API

There are many API drawing functions that *TCanvas* does not encapsulate. Many of these functions are the ones Windows uses to draw standard window elements. Because most of these elements are rectangular, they take *TRect*s as parameters. These functions are hidden gems that most developers don't know about.

We'll start off with the function that Windows uses to draw borders around windows. The *DrawEdge* function can draw either a full window border, or only one of the corner edges. The first parameter is the device context to draw on. The device context of a VCL window is contained in *Canvas.Handle*. The second parameter is a *TRect* defining the border rectangle. The third parameter is a group of flags that specify the type of border to draw, e.g. BDR_RAISEDINNER, EDGE_BUMP, etc. The fourth parameter is a group of flags that specify what the border is, e.g. an edge, a full border, etc. Check the API documentation for *DrawEdge*.

The *DrawFrameControl* function is used by Windows to build almost all the standard controls. This function can draw all the caption buttons (close, maximize, help, minimize, restore, etc.) or check boxes and radio buttons. It can also draw scrollbar buttons in all directions. It can draw the buttons in normal, disabled, or down states. In other words, this is the Windows-control-drawing workhorse. It takes a drawing *TRect* and other parameters that specify the type of button to draw.

ExtTextOut is the big brother of the VCL's *TCanvas.TextRect*, and allows for many more formatting options. It allows you to justify the text as right, left, center, top, or bottom. You can also word-wrap the text in the formatting rectangle.

To draw a line of text surrounded by a three-dimensional box (raised or lowered), you can use the *DrawStatusText* API function. This function was originally intended to allow programs to draw text segments on status bars, but can draw to any other device context.

Conclusion

Some beginning developers tend to shy away from a procedure that takes a record, and would rather stick to procedures that take verbose parameters. They would rather labor over the rectangle method, which takes four integers, than pass a simple *TRect* to the *FrameRect* parameter. In this article, I've tried to show beginners and advanced developers alike that the *TRect* isn't really so frightening, and can even be downright friendly. Knowing how to use the *TRect* is the key to using many API functions. ∆

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\99\AUG\DI9908MA.

Motty Adler is a freelance programmer/consultant and president of WAISS Systems. He has been developing software for over five years, and has used Delphi for over two years. He can be reached at aissoft@aol.com. WAISS Systems' Web site is http://www.waiss.com.

By Jeff Sims

Marotz Cost Xpert

Project Maintenance and Cost Estimating

Software development is a scary subject for too many organizations. One of the biggest problems related to software development projects is estimating how long it will take and how much money it will cost to create and maintain the software. Cost Xpert, developed by Marotz, Inc., is a Delphi-developed tool designed to solve that problem. Marotz offers Cost Xpert as the standard estimating tool for Delphi development.

Typically, developers don't care to talk about due dates, software metrics, lines of code, or productivity. Ask them how long it will take to develop a system and you're lucky if they begin by breaking down the work into measurable, manageable tasks. The organizations that hire software developers apparently behave like their developers. Based on extensive research performed by Marotz on over 8,000 software development projects, "more projects are doomed from poor cost and schedule estimates than ever succumb to technical, political, or development team problems." Some call it our industry's dirty little secret.

With a tool like Cost Xpert, however, you have an important advantage. It can mean the difference between success and failure for you and your project, if not survival and extinction for you and the organization. No kidding! There are many other factors contributing to a successful project, but proper planning, including estimating and scheduling, are critical.

Methodical or scientific estimating includes terms such as SLOC (source lines of code), function points, GUI metrics, object metrics, lifecycles, standards, best case, worst case, expected case, coefficients, metrics, configuration management, etc. It's alright if the vocabulary isn't familiar to you because Marotz has gone a long way to help the user quickly become productive with this very powerful product.

Getting Started

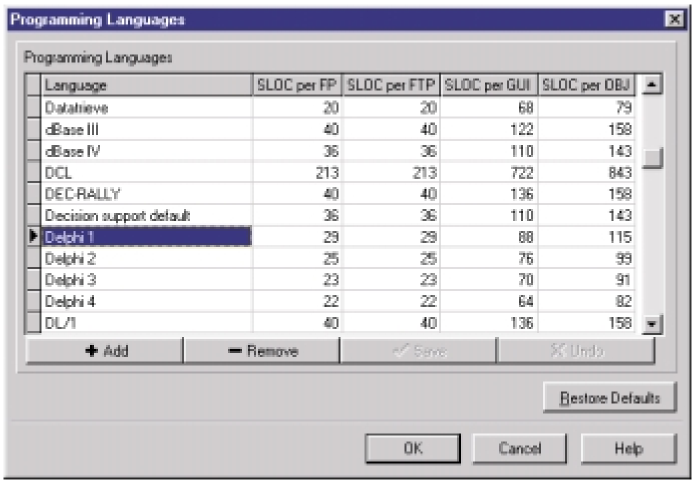
Download an evaluation copy of this product and see how well a tool can be designed using Delphi as the development tool. The source code isn't included, but look at the finished product. Without this elegant interface, I can't seriously imagine someone trying to use a tool that incorporates so much power. The menus, tabs, sub-tabs, grids, and other Windows controls are well organized and arranged for both ease-of-learning and use. The use of fonts, colors, and boxes doesn't distract, but facilitates the user's understanding.

Another reason to download Cost Xpert is to see how easy it can be to install software. And if you get the CD-ROM, you'll also see how nicely software can be packaged. Along with the product, the CD-ROM includes an insightful narrated demonstration, some helpful and relevant articles from TSEPM (*Trends in Software Engineering Process Management*, an electronic journal published by Marotz), and marketing literature on Marotz and Cost Xpert. This material helps the user become productive as soon as possible.

The product is also reliable. It didn't crash once during my evaluation. I've been running an evaluation copy — which is good for 45 days from the day you install it — using the demonstration project, as well as several of my own, and haven't come across any bugs. It stores the project data in a Paradox database (or, should I say a series of Paradox tables in a subdirectory using the BDE?). Cost Xpert will run on a network.

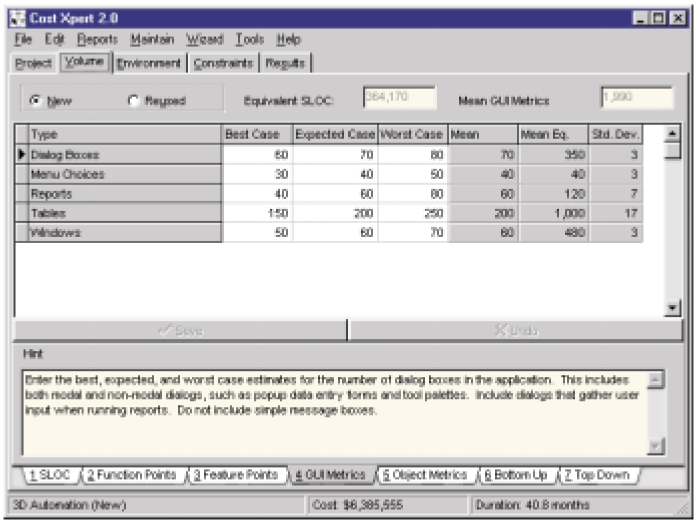
This brings me to a minor complaint. When saving projects, it lets you use the same project name more than once. If you choose to keep more than one version of the same project, I recommend you give it a different name. I will identify additional complaints, but nothing is close to a showstopper.

Based on researching 8,000 projects (commercial, military, and scientific) over a span of 18 years, Marotz has gathered enough data to make the following claim. Cost Xpert "accurately predicts project costs within ± 5 percent given accurate inputs and compares within ± 2 percent to other commercial estimating tools on the market costing $50,000 to $200,000." It's outside the scope of this review for me to confirm or deny Marotz's claims, but I will testify that the estimates produced for me based on my experience (i.e. completed projects) appear to be remarkably accurate.



Language	SLOC per FP	SLOC per FTP	SLOC per GUI	SLOC per OBJ
Datatrieve	20	20	68	79
dBase III	40	40	122	158
dBase IV	36	36	110	143
DCL	213	213	722	843
DEC-RALLY	40	40	136	158
Decision support default	36	36	110	143
Delphi 1	29	29	88	115
Delphi 2	25	25	76	99
Delphi 3	23	23	70	91
Delphi 4	22	22	64	82
DL/1	40	40	136	158

Figure 1: The SLOC (source lines of code) productivity table.



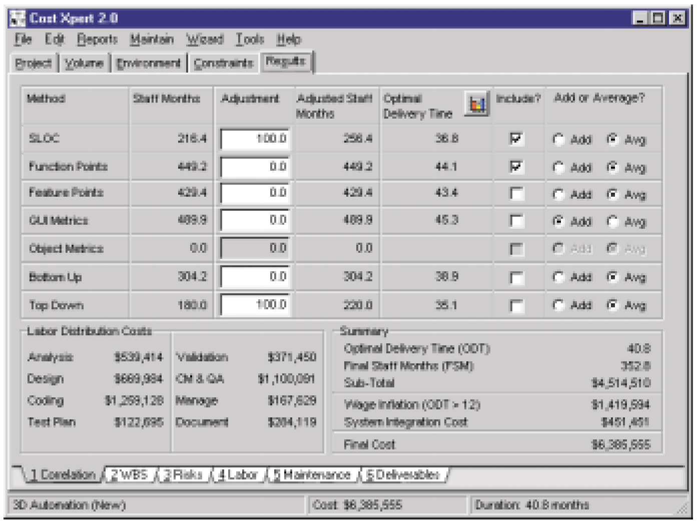
Equivalent SLOC: 364,170 Mean GUI Metrics 1,990

Type	Best Case	Expected Case	Worst Case	Mean	Mean Eq.	Std. Dev.
Dialog Boxes	60	70	80	70	350	3
Menu Choices	30	40	50	40	40	3
Reports	40	60	80	60	120	7
Tables	150	200	250	200	1,000	17
Windows	50	60	70	60	480	3

Enter the best, expected, and worst case estimates for the number of dialog boxes in the application. This includes both modal and non-modal dialogs, such as popup data entry forms and tool palettes. Include dialogs that gather user input when running reports. Do not include simple message boxes.

3D Automation (New) Cost: $6,385,555 Duration: 40.8 months

Figure 2: An estimation of number of functions, features, modules, or objects.


Cost Xpert 2.0
File Edit Reports Maintain Wizard Tools Help
Project Volume Environment Constraints Results
1 Correlation
2 WBS
3 Risks
4 Labor
5 Maintenance
6 Deliverables

Method	Staff Months	Adjustment	Adjusted Staff Months	Optimal Delivery Time	Include?	Add or Average?
SLOC	216.4	100.0	256.4	36.8	✓	Add [✓ Avg]
Function Points	449.2	0.0	449.2	44.1	✓	Add [✓ Avg]
Feature Points	429.4	0.0	429.4	43.4		Add [✓ Avg]
GUI Metrics	489.9	0.0	489.9	45.3		[✓ Add] Avg
Object Metrics	0.0	0.0	0.0			Add [✓ Avg]
Bottom Up	304.2	0.0	304.2	38.9		Add [✓ Avg]
Top Down	180.0	100.0	220.0	35.1		Add [✓ Avg]

Labor Distribution Costs

Analysis	$539,414	Validation	$371,450
Design	$669,984	CM & QA	$1,100,091
Coding	$1,259,128	Manage	$167,629
Test Plan	$122,695	Document	$204,119

Summary

Optimal Delivery Time (ODT)	40.8
Final Staff Months (FSM)	352.8
Sub-Total	$4,514,510
Wage Inflation (ODT > 12)	$1,419,594
System Integration Cost	$451,451
Final Cost	$6,385,555

3D Automation (New) Cost: $6,385,555 Duration: 40.8 months

Figure 3: The sum or average of varying approaches gives an accurate result.

A Look Inside the Engine

Cost Xpert supports estimates for over 500 programming languages, including Delphi versions 1 through 4, and every programming language I could imagine (even Paradox!). However, for any project, only two languages may be included. The estimator must specify a percentage when two languages are included. This limitation bothered me initially because many projects seem to include more than two languages (e.g. Delphi, SQL, Java, a report writer, etc.). One solution is to create multiple project estimates. Unfortunately, this could become an administrative problem, especially because you can and may want to output the Cost Xpert results and import that data into Microsoft Project for managing the project, and I doubt you would want to administer two projects. However, when you look up the productivity assumptions Marotz includes for each language, you might find that you're better off grouping the third or fourth language along with one of the first two (based on the SLOC productivity).

Access the programming language maintenance function (i.e. the SLOC productivity table) and you'll see that the Delphi development community has enjoyed increasing productivity with each subsequent release of Delphi (see Figure 1). Using GUI metrics (where dialog boxes, menu choices, reports, tables, and windows are how modules are defined), the relative number of SLOC per module has dropped from 88 to 76, 70, and 64 for each subsequent release of Delphi. Contrast this with C++ = 198, C++Builder = 95, Cobol = 399, Paradox/PAL = 110, and SQL = 48. This information alone is extremely valuable. If someone in the organization wants to specify C++ as the development language (instead of Delphi, for example), then an objective and unemotional comparison will clearly and tangibly explain the difference in time and money.

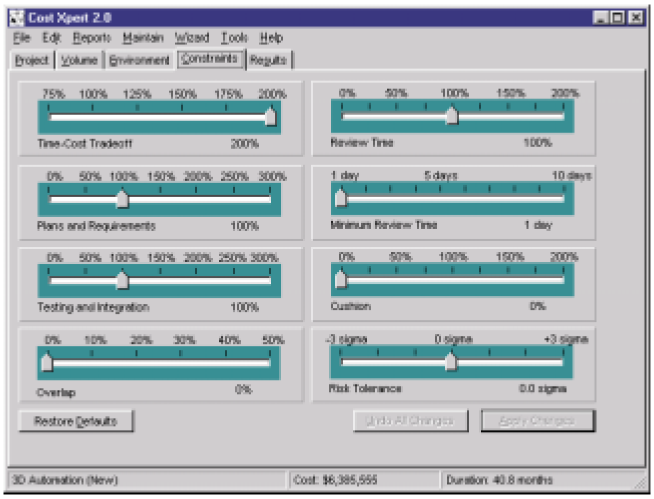
Unfortunately, there are no "complex" or "simple" functions, features, modules, or objects. Although a dialog box and a menu choice, for example, have different metric equivalent values, Cost Xpert doesn't distinguish between a complex window module, which might require three times the code, and a simple window module. Alternatively, complex modules could be singled out and specified in the SLOC or feature points (under algorithms, for example), then added to the GUI metric for a total project estimate.

Practically everything is accessible to the user through the maintenance menu, with support to add and modify the parameter values (project types, lifecycles, languages, etc.). If this sounds complicated, it could be because the product has tremendous flexibility. It's more likely because I only have 2,000 words to explain this powerful product. Download an evaluation copy and see for yourself. A demonstration is worth a million words.

Although the product uses industry-standard terms and has designed the product for ease-of-learning and use, the product still assumes the estimator has experience in estimating software development projects. For instance, the estimator needs to be able to recognize whether a project will require more or less than the industry standard of 18 percent for system integration.

Volumes

At the heart of an estimate is the number of functions, features, modules, or objects (see Figure 2). The number of SLOC is then calculated (alternatively, the estimated SLOC can be entered directly). There are seven estimating approaches supported: SLOC, function points, feature points, GUI metrics,


Cost Xpert 2.0
File Edit Reports Maintain Wizard Tools Help
Project Volume Environment Constraints Results
Restore Defaults
Undo All Changes
Apply Changes

Time-Cost Tradeoff 200%
Review Time 100%
Plans and Requirements 100%
Minimum Review Time 1 day
Testing and Integration 100%
Cushion 0%
Overlap 0%
Risk Tolerance 0.0 sigma

3D Automation (New) | Cost: $6,385,555 | Duration: 40.8 months

Figure 4: Manipulating project-level parameters.

object metrics, bottom up, and top down. It also supports and distinguishes between new and reused code. And projects can be the sum or average of more than one of the different approaches (see Figure 3). This is valuable as different parts of a project are better estimated using one approach over another. A project doesn't have to be just one approach.

Fine Tuning the Estimate

The Constraints page includes eight sliding controls that manipulate a variety of project-level parameters (see Figure 4). The **Time-Cost Tradeoff**, for example, is how you can speed up the delivery while increasing the cost, or reduce the cost while taking more time. Unfortunately, there's no way to specifically limit the number of staff available during specific months. It's only this indirect slider that increases the length of the project by 200 percent and reduces peak staffing levels. We all know that one of the easiest ways to shorten a software schedule is to increase the size of the team. But this usually makes each person less efficient. The ability to limit the size of the project team and maximize per-person productivity is very important, but simply isn't supported in Cost Xpert.

Pros and Cons

Every product has its ups and downs. Cost Xpert is no exception. Below are some of the good and bad features of the product.

Pros
- Supports seven estimating approaches
- Supports CMM (Capability Maturity Model)
- Support for COCOMO II (Constructive Cost Model) estimating methodology
- Support for over 500 programming languages (with ability to add/change)
- Supports military, MIS, systems, and Y2K project types (with ability to add/change)
- Elegant design, ease-of-learning and use
- Excellent supplemental material included on CD-ROM
- Well-suited to software development methodologies that produce deliverables in a "waterfall"
- Parameter values are user maintainable
- Output reports to printer and RTF files
- Export estimates/baseline plans to Microsoft Project for project management

Cons
- No explicit support for RAD, iterative, or rapid prototyping methodologies — RAD and 4GL languages are supported, and additional life cycles (e.g. iterative) can be added by the user
- Limited to two languages per project
- All functions, features, modules, or objects are assumed to be of equal complexity (vs. simple, medium, and complex functions, for example)
- Limited control over staffing levels

— *Jeff Sims*

Documentation

Part of practically all software development projects today is the requirement to include documentation. Even if it's "shelfware," printed documentation is tangible. More importantly, users can see, study, and understand what the analyst has documented as requirements in diagrams and words. Cost Xpert includes the standard document types, e.g. software development plan, software requirement specification, system architecture specification, general system design, detailed system design, software test plan, etc. Based on the volumes input, Cost Xpert estimates the number of pages that will be required for each document. The user can include or exclude any one of the documents.

Interestingly, excluding a document will result in a reduced estimate. This may seem logical on the surface, but it begs the question: "Why document anything?" The answer is because if you don't document the system, you're taking a risk. Not documenting the system should actually increase the estimate, because users are more likely to be dissatisfied with the finished product. This will result in revisions that would not have been necessary if users had seen the discrepancy earlier (i.e. in a design document), when the cost to repair the problem was much lower. For example, reworking a software-requirements problem once the software is in operation typically costs 50 to 200 times what it would take to rework the problem in the requirements stage.

INFORMANT FACT FILE

Cost Xpert is much like hiring an MBA to help you add the discipline of estimating your project. It's likely to add the new dimensions of predictability and control to your organization. Easy installation and collateral material — including a CD-ROM containing a narrated demonstration, some helpful and relevant articles from Marotz's *Trends in Software Engineering Process Management* journal, and marketing literature — adds greater usability to the product. I suggest you download it now.

Marotz, Inc.
13518 Jamul Dr.
Jamul, CA 91935

Phone: (800) 477-6168 or (619) 660-2010
E-Mail: sales@marotz.com
Web Site: http://www.marotz.com
Price: US$1,995

Conclusion

Cost Xpert is much like hiring an MBA to help you add the discipline of estimating your project. Like any powerful tool, there is a risk of misuse: You might be tempted to use the wizards and fill in the blanks mindlessly and irresponsibly. If you don't question estimates and closely review all assumptions and work with the product diligently, you may find that your estimates are ridiculously high and the projects unaffordable. On the other hand, Cost Xpert is likely to add the new dimensions of predictability and control to your organization. When someone wants to change the project scope, you can go back and estimate the impact. Instead of not knowing what to say to your client/user, you can calmly explain how much longer and how much more money the project will now require. I recommend it, and suggest you download it and see for yourself. ∆

References

- Hetzel, Bill. *Making Software Measurement Work: Building an Effective Measurement Program*. New York: John Wiley & Sons, 1993.
- Boehm, Barry W., and Philip N. Papaccio. "Understanding and Controlling Software Costs." *IEEE Transactions on Software Engineering*, SE-15 (July, 1988): 902-916.

Jeff Sims is a Northern California-based independent software consultant who specializes in data and process modeling, business intelligence, project planning, and project management for Fortune 500 and government clients. He can be reached at jeff@metagraph.com, or (650) 359-7851.

By Warren Rachele

Advantage Database Server 5.1

Safe Shores for Client/Server Programmers

Scalability, thin client, and increased performance are all part of the siren song of client/server database programming. The allure of these and other genre features tempt database developers attempting to steer their ships to safety. Getting there, however, is a complex, arduous process. And just as the mythological sirens led sailors to a rocky doom, client/server programming can lead to trouble without the aid of a well-designed and implemented server and engine product.

The Advantage Database Server from Extended Systems, Inc. provides safe shores for developers in search of a client/server solution. The core component systems of the product, available in a number of language- and protocol-specific configurations, comprise two pieces. Primary is the Advantage Database Server, which resides on a file server accessible by all your client applications. Complementing this tool are the client-side APIs and visual controls that implement the communications with the server and handle database transactions. Access to the database is BDE-free, talking directly to the database server for all transactions, allowing your application to do away with the overhead associated with the Borland Database Engine.

The Advantage Database Server is a high-performance database engine that will run on a Windows NT or Novell NetWare file server. It supports an Xbase family of file structures: dBASE III, FoxPro, Clipper, and Extended's proprietary file format. When initiated and running on the server side, all database transactions are handled by the Advantage Server. The developer coming from other environments, such as FoxPro or the BDE, will find this to be a monumental change.

A networked database application that doesn't use a database server, such as a BDE-based project, places the tables and indices that make up the application into a shared folder on the file server. By necessity, all users of the application have permissions and access to this folder. When data requests are made of this share, all the data is sent to the requesting desktop for processing, resulting in an enormous amount of network traffic and the possibility of concurrency issues. A database server takes a different approach; when your program makes a data request, the database server accesses the files and the data is processed on the file server rather than being sent over the wire to your desktop. Instead of the raw data, your application sees the results of the operation, whether it be the display of a result set or a modification to the records of a table.

Using a database server, the client application is freed from management issues, such as file or record locking, and concurrency. Network traffic is reduced by a large factor, increasing the overall performance of all network services. Integrity and performance specific to the database are also vastly improved. The Advantage Database implements the integrity operations of the relational database specification, ensuring that all updates run to completion, i.e. it supports transaction processing.

The requirements of the client side of the equation are also changed in a client/server system. In the traditional multi-user database application, the client is responsible for data retrieval from the share, and then the processing of the data locally. These requirements build a lot of overhead into your application in the form of API libraries and support DLLs that must be distributed with the application. The Advantage client libraries reduce this overhead significantly.

The Advantage Database Server

Extended Systems ships the complete Advantage solution in two parts: the Advantage Database Server and the Client Engine Kits. For this review, I selected the latest version of the Advantage Database Server for Windows NT, version 5.1 (see sidebar "Advantage Database Server 5.5" for information on the latest release). The target machine was a Pentium II-based box with 128MB of memory using Windows NT Server 4.0. The server installed and initiated flawlessly.

The Advantage Server takes less than 2MB of disk space, and runs as a Windows NT service. Services are run in the background on the operating system and usually have no user interface. Such is the case with the Advantage Server. Through the Task Manager, it's easy to gather information about running services, and it showed that the Advantage Server takes about 4MB of memory to run. The service can be configured to run automatically when the server is booted or started manually.

The service configuration for this application makes the database management more robust and secure, especially when configured to start automatically. If anything happens to the NT server, and a reboot is required, an automatic startup for the service guarantees accessibility to the database clients without requiring manual intervention on the part of the NT or database administrator.

Configuring the Advantage Server for maximum performance is simple, with one caveat. The developer and NT Administrator must have an understanding of the requirements and limitations of the Advantage environment before making configuration changes. There are a number of differences between it and the commonly used Paradox tools, and these differences come into play in setting the configuration parameters for the server. There are two ways of setting these parameters: on the command line or through the Advantage Configuration Utility. Configuration settings are stored in the Windows registry and require the service be stopped and restarted to take effect.

The Advantage Server supports a number of ISAM (Index Sequential Access Method) file formats. While the server continues to support the DBF file formats from CA-Clipper, dBASE III+, and Microsoft FoxPro, new in version 5.1 is support for the proprietary ADT table format. This higher-performance format is an extension of the DBF structure and includes additional features, such as:

- long field names,
- new field data types,
- true unique indexes for primary key support,
- deleted record re-use,
- an increase in maximum number of records and a larger file size, and
- more secure encryption.

The ADT table and associated ADI indexes and ADM memos are the preferred structure for development of new applications because they offer the highest performance and a familiar format.

The Advantage Client Engine

The other half of the client/server equation is, of course, the client. Extended Systems markets client kits for Delphi, CA-Clipper, FoxPro, and CA-Visual Objects, as well as drivers for ODBC clients. All the client kits intended for development in the Windows environment use the Advantage Client Engine for access to the Advantage Server. In addition to server access, the Client Engine provides a large measure of scalability, allowing the developer to create a local database application that doesn't require the Advantage Server. After testing, or as needed, the application can simply be scaled up by pointing the project to the remote server; no other changes are necessary to the code.

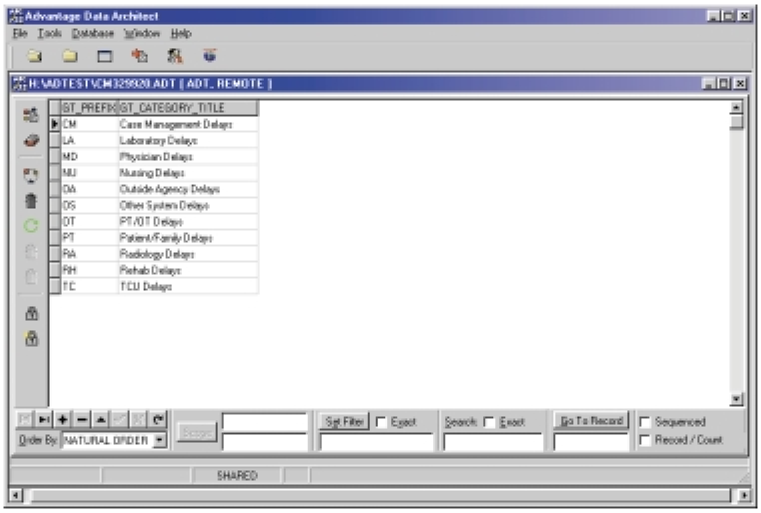
The Advantage Database Engine for Delphi provides the developer with a complete set of solutions for accessing the Advantage Server. The programmer can use the BDE Alternative, a *TDataSet* descendant that replaces the Table component, or access the server directly through the Advantage Client Engine API with the Advantage Client Engine SDK. Although they contain overlapping functionality, each of these options targets a specific development activity. The BDE Alternative replaces the BDE DLLs that can eliminate the need for the BDE, or allows BDE-specific calls, such as access to a Paradox table, to pass through to the BDE. The advantage of this model is that projects can be quickly moved to the Advantage environment, or a mixture of BDE and Advantage tools, Paradox and Advantage tables for example, can be used together in the same project. The BDE Alternative allows the developer to use the Table component, or Advantage's AdsTable component for its extended functionality.

The preferred development tool when implementing a true client/server solution using the Advantage tools is the *TDataSet* descendant. This set of native Delphi components simplifies the connectivity to the Advantage Server. The key to these components is the AdsTable control that parallels the Table component, and works with any native or third-party components that require a *TDataSet* descendant. By using the AdsTable component, the application gains access to the extended functionality of the Advantage systems; nearly all the methods and properties of the standard Table are exposed while adding accessing to Advantage-specific properties.

Much as the BDE API allows the developer direct access to the low-level functions of the Database Engine, the Advantage Client SDK includes the PAS files necessary to bypass the need to use the visual controls. Integrating these files into your Delphi application gives you complete control over access to the Advantage Server. As with any hand-coded solution, the developer must carefully consider the balance of the additional work to the increased functionality to determine if the required effort is worthwhile.

Installation and Configuration

Installation of the Advantage Database Engine for Delphi is straightforward. A package file installs itself automatically into the Delphi IDE without problem. A new Advantage page on the Component palette is created that contains the three visual components: AdsTable (that's been mentioned), AdsSettings, and AdsConnection. The AdsSettings component encapsulates Advantage-specific parameters that affect all tables opened in the project. This component allows your program to control aspects of the database, such as whether deleted records should be shown, and whether the local, Internet, or remote server should be used to access the data tables. The AdsConnection component is available for integrating Advantage



ST_PREFIX	ST_CATEGORY_TITLE
CM	Case Management Delays
LA	Laboratory Delays
MD	Physician Delays
NU	Nursing Delays
OA	Outside Agency Delays
OS	Other System Delays
OT	PT/OT Delays
PT	Patient/Family Delays
RA	Radiology Delays
RH	Rehab Delays
TC	TCU Delays

Figure 1: The Advantage Data Architect is similar to the Borland Database Desktop in format and usage.

Advantage Database Server 5.5

Already a strong performer in the PC client/server database market, the Advantage Database Server and Client kits receive a major upgrade with the version 5.5 release. At press time, this upgrade was due to be released in early to mid-summer and looks like a must-have for any Advantage developer. The developers at Extended Systems have followed their success with the Database Engine for Delphi by extending the functionality of the product into the SQL arena.

The Advantage Database Server gains a SQL engine that supports a subset of the ANSI-92 SQL specification. These new relational capabilities are tightly integrated with Advantage's existing functionality and offer a high measure of data integrity. The new functions apply the same design already familiar to Advantage developers, making it easy to implement the new functions into existing applications.

This functionality is delivered via a new component named AdsQuery. As with the AdsTable control, the new query component is a parallel object to Delphi's native Query component. This new control is descended from *TDataSet* and allows developers to submit queries and other SQL commands through a familiar interface. If a result set is returned, the cursor handle can be used to navigate the dataset in a table-like fashion, row-by-row — a feature missing from many SQL implementations.

One of the benefits of the Advantage client/server system is its ability to easily scale up or down. This gives the developer the opportunity to build and test an application locally, using the full feature set. When the roll-out date arrives, the application is simply pointed to the file server running the Advantage Database Server to realize a fully functioning client/server application. This scalability benefit remains in place with the new components and API functions.

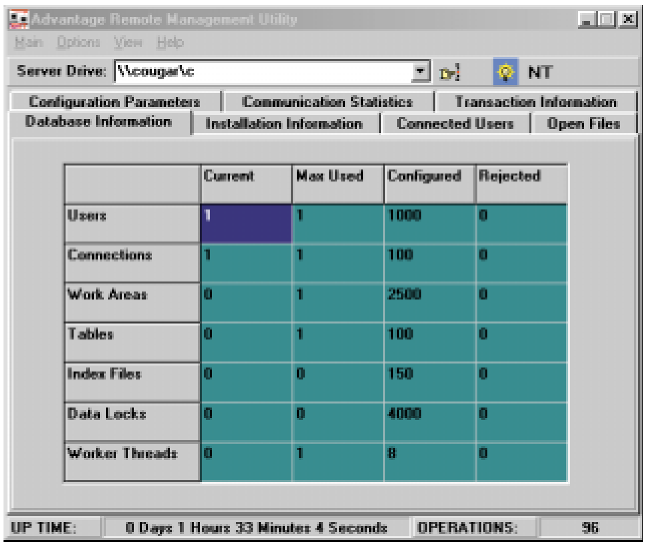
The 5.5 release includes a more robust encryption function, making security even better within the Advantage tool set. Other minor improvements, such as Memo recycling, which makes more efficient use of file space, are also realized in this incremental release.

— *Warren Rachele*

Transaction Processing into an application and controlling server connections.

There are a number of peripheral utilities used with the Advantage client/server system. One of the more important is the Advantage Data Architect. As shown in Figure 1, this utility is similar in usage and format to the Borland Database Desktop. It can be used to build and modify tables in a variety of structures, but its primary use is to create tables in the Advantage ADT file format. Additionally, the utility can perform data conversions from Paradox/dBASE, FoxPro, or Clipper tables to the Advantage proprietary structure to take advantage of the performance and functionality extensions of the format. Index and table maintenance, filtering, sorting, and other table- and database-specific functions can be accessed through the Advantage Data Architect tool.

Another utility that comes into play is the Advantage Management Utility, which is used to access information about the Advantage Database Server. A deceptively simple interface gives the developer the ability to monitor server-side activity from the desktop. Figure 2 shows the Remote Management Utility in action with the Tabbed display view selected. This gives the user a quick way of reviewing server statistics. Information available through this interface includes server type and up time, installation and configuration of the database server itself, open files, communications statistics, and transaction information.

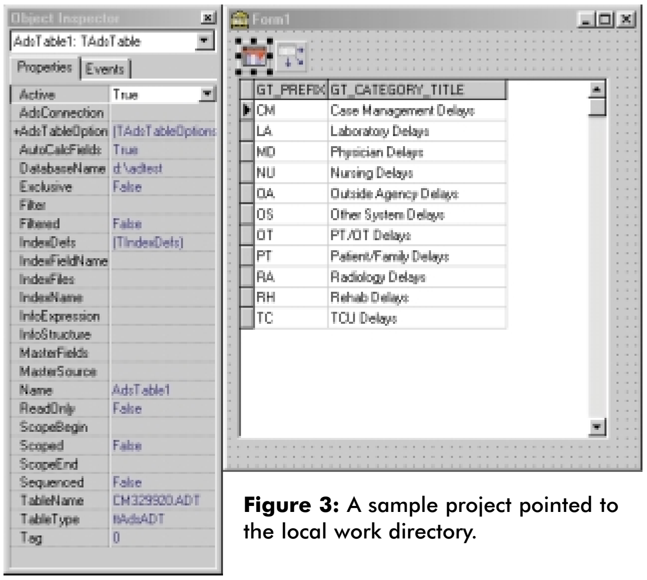


Server Drive: \\cougar\c

	Current	Max Used	Configured	Rejected
Users	1	1	1000	0
Connections	1	1	100	0
Work Areas	0	1	2500	0
Tables	0	1	100	0
Index Files	0	0	150	0
Data Locks	0	0	4000	0
Worker Threads	0	1	8	0

UP TIME: 0 Days 1 Hours 33 Minutes 4 Seconds OPERATIONS: 96

Figure 2: The Advantage Remote Management Utility with the Tabbed display view selected.



AdsTable1: TAdsTable

Active	True
AdsConnection	
+AdsTableOption	(TAdsTableOptions
AutoCalcFields	True
DatabaseName	d:\adtest
Exclusive	False
Filter	
Filtered	False
IndexDefs	(TIndexDefs)
IndexFieldName	
IndexFiles	
IndexName	
InfoExpression	
InfoStructure	
MasterFields	
MasterSource	
Name	AdsTable1
ReadOnly	False
ScopeBegin	
Scoped	False
ScopeEnd	
Sequenced	False
TableName	CM329920.ADT
TableType	ttAdsADT
Tag	0

GT_PREFIX	GT_CATEGORY_TITLE
CM	Case Management Delays
LA	Laboratory Delays
MD	Physician Delays
NU	Nursing Delays
OA	Outside Agency Delays
OS	Other System Delays
OT	PT/OT Delays
PT	Patient/Family Delays
RA	Radiology Delays
RH	Rehab Delays
TC	TCU Delays

Figure 3: A sample project pointed to the local work directory.

Using the Advantage Database System

The complexities of a client/server application are nicely encapsulated by the Advantage system. A developer can easily take advantage of the scalability designed into this system by starting development locally. A simple project consisting of a form, an AdsTable, a DataSource, and a DBGrid is all that's needed to demonstrate this faculty. Adding the AdsTable to the form, point the *DatabaseName* property to a local directory for development purposes. The full set of capabilities for the Advantage Server are present in the Local Server services. Figure 3 is a sample project that is pointed to the local work directory. The project can be fully developed and tested at the local level, then simply scaled up by modifying the database name to point to a network share recognized by the Advantage Server.

Obviously, such a simple exercise is a trivial examination of the capabilities of the Advantage system. Equally likely to occur on the project continuum is the conversion of an existing single- or multi-user database to a client/server application. By descending the main data access object from a DataSet component, nearly every method and property is paralleled in the AdsTable and Table components. This design makes it a simple matter to substitute the Advantage control on a one-to-one basis for each incidence of the Table component, without having to change any other parts of your application.

As a test of how transparent this process could be, I took an existing Paradox-based application and attempted to convert the entire project to an Advantage Server-based program. Before jumping into the Delphi environment, I paralleled the database on the server as an Advantage proprietary ADT database. This process was easily completed through the Advantage Data Architect Utility. Selecting the Import option, I provided the target and destination directory and share as appropriate, then the needed data type information. Clicking on Execute completed the process, giving me an entirely new database on my server.

The next step in this conversion process was to modify the application itself. Depending on the original design of the application, this may be a

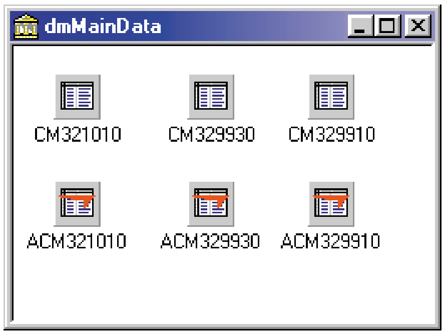


Figure 4: The Data Module.

complex or trivial process. In the project that I chose, the data tables are contained within a Data Module, centralizing and limiting the number of parallel tables that needed to be created. Within the Data Module, shown in Figure 4, I started by adding AdsTable components on a one-to-one basis for each Table component in the module. Being cautious, I converted one at a time and tested the application in between. The new AdsTable components were named in a similar fashion, and the *DatabaseName* property was pointed to the server share. The connection was made without a hitch, and when I clicked on the *TableName* property to specify the table, I was greeted by a list of all the converted objects. Having done that, I made the table active, and proceeded to test the application. The process completed without incident. In fact, the application was able to run with both the Advantage Server table and the remaining BDE-based tables active during the testing process, demonstrating that the two types of tables, ADT and Paradox, are able to co-exist. Developers hesitant to consider moving to the Advantage Database Server might reconsider, given the ease of the transition with their current and production projects. After the initial transition of getting the application to work with the Database server, the developer can then consider utilizing the advanced features exposed by the Advantage tools.

Transaction Processing is one of those advanced features most likely to catch a developer's attention. Transaction processing is built around the concept that a database should always be able to maintain a specific state, regardless of the status of operations occurring upon it. In other words, all insert, update, or delete operations will be completed, or no part of the operation will occur. This prevents the database from reflecting partially successful updates and destroying the integrity of the data. While it's possible for this corruption to occur in a single-user environment, it has an even greater probability when the database is accessed through a network connection. The connectivity alone introduces a number of points of failure into the equation, making the protection of the data integrity even more critical. With a full pallet of commit and rollback features, the Advantage Transaction Processing System brings this critical functionality to the Xbase development arena.

Conclusion

Some Delphi developers already have a client/server database subset at their disposal through the InterBase tools and components included with their development tool. To fully implement this product is an expensive and complex process with a long learning curve. The Advantage Database Server, on the other hand, is built around the familiar at a far lower cost of entry. Developers immersed in the Xbase school of development will recognize the parallel concepts immediately, and therefore, have a leg up on learning the advanced features. Delphi developers will be comfortable with the components, especially the immediate usefulness of AdsTable.

On the other hand, the Advantage Database Server is made unnecessarily complex by a lack of clear and immediately available documentation. The printed manuals that come with the Server and Client packages are easily dismissed because they provide little in the way of implementation information. For that, the developer is referred to examples that exist only on the Extended Systems Web site, or he or she must delve into the online documentation. Online documentation works best in situations where the user already knows what they are looking for, and can proceed through the layers of links in a linear fashion to locate it. If you're looking for an item, however, on which you and the documentation writer do not share similar vocabulary, you'll end up wading through endless Help screens and abandoning the process in frustration.

Getting started with a new product such as the Advantage Server and Client kits should not be this way. Clearly printed documentation should, at a minimum, be included that makes your first experiences with the product a success. An example of how this would save enormous time comes immediately to mind. The BDE developer will immediately think of an Alias when setting the database name in the AdsTable component. Without clear instructions explaining that this is not the case with the Advantage product, the developer will waste an enormous amount of time searching for this answer. Another question of this type is simply how to create an ADT table. You must dig deep to identify the Advantage Data Architect as the solution to this question. Considerations such as this are often difficult for a company to see when all documentation is developed in-house. Familiarity with the product sometimes takes the focus away from the user's perspective, where it belongs.

INFORMANT FACT FILE

With the lower cost and system requirements for the Advantage Database Server, and the benefits of the Advantage Database Engine for Delphi on the client side, this product is an excellent choice for programmers who wish to move up to client-server functionality. It works well, integrates easily into existing projects, and offers a number of advanced features that make it more desirable than competing tools. Although the Advantage Database Server is made unnecessarily complex by a lack of clear and immediately available documentation, its performance is exemplary.

Extended Systems, Inc.
5777 N. Meeker Ave.
Boise, ID 83713

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

The performance of the Advantage Database Server is exemplary. Although none of the test projects came close to testing the outer boundaries of the capability of the product, it performed without complaint, no matter how flawed my commands might have been. Any errors that occurred were fully documented in the included Help files and were easily corrected. Installation and startup of the server was perfect and, aside from some upgrade problems of my own making, the installation of the Delphi side of things went without problems as well.

With the lower cost and system requirements for the Advantage Database Server, and the benefits of the Advantage Database Engine for Delphi on the client side, this product is an excellent choice for programmers wishing to move up to client/server functionality. It works well, integrates easily into existing projects, and offers a number of advanced features that make it more desirable than competing tools. Δ

Warren Rachele is Chief Architect of The Hunter Group, an Evergreen, CO software development company specializing in database-management software. The company has served its customers since 1987. Warren also teaches programming, hardware architecture, and database management at the college level. He can be reached by e-mail at wrachele@earthlink.net.

table of CONTENTS

The Delphi Bookshelf: Multimedia Sources

In the March, 1999 issue of *Delphi Informant*, I presented an overview of the Windows Multimedia APIs. I wrote that, although these APIs have been a part of Delphi since the beginning, their documentation leaves a lot to be desired. It's true that every function and structure is described in the Help file. The problem comes when you try to put all those functions together in a working application. I also indicated in that column that I'm writing a book on this topic, *The Tomes of Delphi, 32-bit Multimedia Programming*, which will be published by Wordware this summer (1999). In the process of writing this book, I have relied on various resources, including books and Web sites. This month, I'd like to provide an overview of some of the books I found most helpful; one is Delphi-specific, the others are for C/C++ developers.

Delphi 2 Multimedia Adventure Set by Scott Jarol, Dan Haygood, and Chris Coppola [Coriolis Group Books, 1996] is the only Delphi book I'm aware of that deals in any comprehensive way with the topic of multimedia. One of the interesting features of this book is the HTML application (Web Browser and Hypertext Engine) that constitutes one of its major threads. This aspect has been controversial: Some developers love it; others feel HTML shouldn't be in a multimedia book. I found the sections on the multimedia APIs very helpful. These include excellent (though incomplete) introductions to the WAVE API, the MIDI API, and MCI (Media Control Interface). These chapters are particularly helpful in providing a context in which to use the functions in mmsystem.pas.

Part of the Waite Group's three-volume *Win32 Bible* series edited by Richard J. Simon, *Windows 95 Multimedia & ODBC API Bible* [Waite Group Press, 1996] presents a thorough overview of Windows' current support for multimedia, telephony, and Open Database Connectivity (ODBC). The multimedia chapters comprise nearly half the book, and provide a comprehensive exposition of all the constants, structures, and functions in the Windows multimedia APIs. This book is indispensable. The explanations are clear and concise, and the C examples provided excellent models for my own example applications (some of which you'll be seeing in *Delphi Informant* in coming months).

Steve Rimmer's *Advanced Multimedia Programming* [Windcrest Books, 1994] is another useful C-oriented book. Rather than concentrating exclusively on the Windows multimedia APIs, this work deals with other techniques often associated with multimedia, including animation. It also includes sections on programming joysticks and .AVI video files, topics not always included in multimedia works. I found the sections on WAVE files and MIDI particularly helpful. However, both topics are so complex that one could write an entire book about each. Which brings us to our last two books.

A Programmer's Guide to Sound by Tim Kientzle [Addison-Wesley, 1997] is a real gem. This book is essential if you're working with multimedia on multiple platforms (particularly Windows, Macintosh, and UNIX). Delphi developers who have a background in C++ will appreciate Kientzle's object-oriented approach, building many small classes that become the basis of larger, more useful classes. The introductory material on sound is excellent and will prove very helpful to developers who don't have a strong background in acoustics and psycho-acoustics. As you would guess, the sections on Waveform audio and MIDI are excellent. However, this book goes further, effectively introducing some lesser-known audio file types.

Last, but certainly not least, Paul Messick's *Maximum MIDI* [Manning Publications Co., 1997] is a seminal work on programming MIDI. In many ways, this may be the most helpful book of all if you're looking for MIDI tools you can easily incorporate into your applications. In addition to providing a comprehensive overview of MIDI, *Maximum MIDI* develops a library of MIDI routines, the MaxMidi.dll (in both 16- and 32-bit formats), with all the source code included. Messick also develops MIDI informational tools and a MIDI sequencer.

In addition to my work with multimedia, I have researched telephony, as well as other forms of communication under Windows. In a future column, I'll discuss some of the resources I found helpful in those ventures, including the Internet, which provides a wealth of resources. As always, I look forward to hearing from you with your suggestions and information. Δ

— 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.