Cover Art By: *Tom McKeith*

Interfacing with the Leading Report Writer

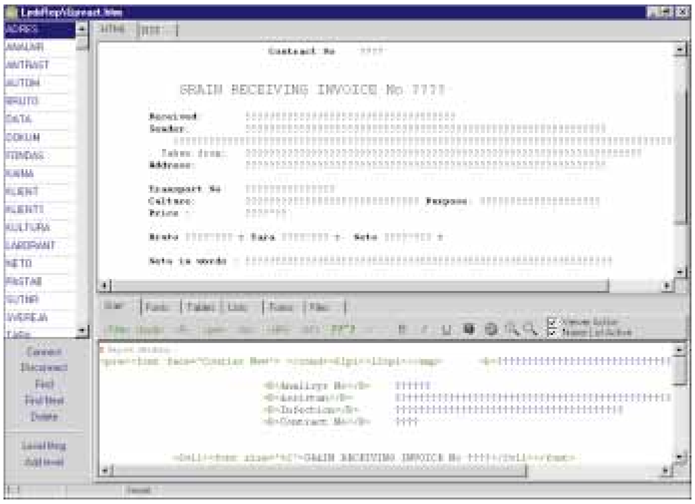
October 1998, Volume 4, Number 10

magazine MENU

Baltic Solutions Releases HTMLReport 2.0

Baltic Solutions released *HTMLReport 2.0*, which generates HTML reports in a Delphi program. HTMLReport 2.0 generates reports not only from BDE-compliant databases, but also from abstract sources, including plain text file databases or a set of Pascal variables.

With HTMLReport 2.0, you can send HTML reports by e-mail and view them with an HTML browser, or transform them into text format and print a draft. You can generate a multi-level report and view it with a built-in HTML browser, and view source data from HTMLReport by clicking on an HTML link. You can also use your favorite HTML editor or viewer to design or view a report.



Baltic Solutions
Price: Standard (without source), US$60; Professional (with source), US$136.
Web Site: http://www.balticsolutions.com

Chant Offers SpeechKit 2

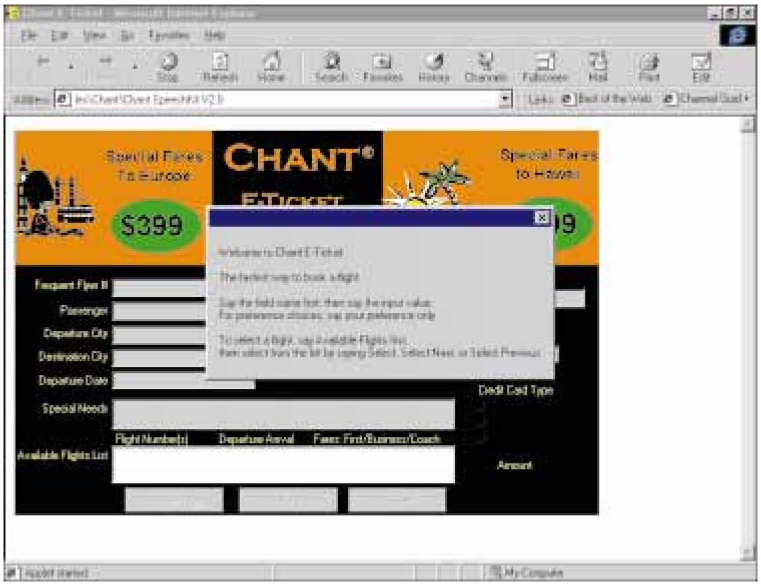
Chant Inc. announced the availability of *SpeechKit 2*, component software for adding speech recognition and speech synthesis (text-to-speech) capabilities to desktop and Web-enabled applications. With their voices, users can fill out forms, query databases, enter transactions, navigate applications, ask questions, and get responses without using a mouse or keyboard. SpeechKit 2 manages vocabularies and places recognized utterances directly in form fields, query windows, or transaction windows.

SpeechKit 2 enables developers to build speech-aware applications with minimal programming. SpeechKit 2 isolates applications from the complexities of speech engine APIs, managing vocabularies, and displaying optional visual cues. The component software eliminates the need for application code to access low-level APIs, create and manage data structures, and handle OS events.

SpeechKit 2 works with applications developed using Delphi, Visual C++, Visual Basic, Visual J++, Visual FoxPro, Smalltalk, C++Builder, and JBuilder.

SpeechKit 2 works with speech recognition and text-to-speech engines that support Microsoft's Speech API and IBM's Speech Manager API.

SpeechKit 2 also supports ANSI single-byte, multi-byte, and Unicode characters that enable applications to support multiple languages and locales.



Chant Inc.
Price: US$399 per developer, no royalties; SpeechKit 2 is available in ActiveX, Java JNI, and C/C++ DLL forms.
Phone: (888) 8CHANT8
Web Site: http://www.chant.net

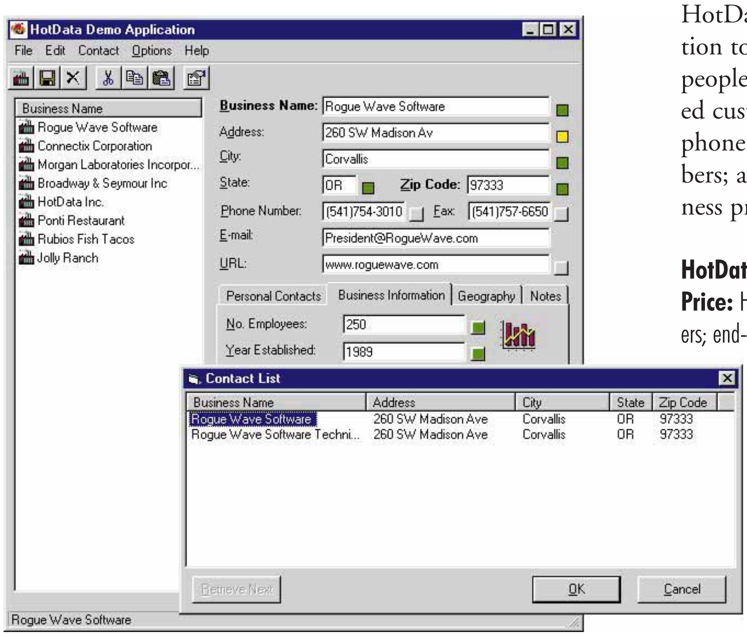
HotData Launches Internet Service for Automated Data Access

HotData, Inc. announced the release of ***HotData 1.0***, a service that delivers data items directly into desktop software applications via the Internet.

Software developers can embed HotData functionality into new or existing applications using the HotData Developer's Kit (HDK). It supports Windows 95/98/NT development environments, including Delphi, C/C++, PowerBuilder, Visual Basic, and Excel, and is available to qualified developers at no charge.

Once enabled with HotData, applications connect via the Internet to the HotData system, an online data clearinghouse containing a variety of data items, such as contact information, phone and fax numbers, credit ratings, SIC codes, CASS, and demographics. The data is provided by a group of data vendors.

The company has targeted developers of sales force automation applications because the market segment is best suited for the company's existing data sets. HotData provides information to save time for salespeople searching for relocated customers; finding telephone, e-mail, and fax numbers; and looking up business profiles.


HotData Demo Application

File Edit Contact Options Help

Business Name
Rogue Wave Software
Connectix Corporation
Morgan Laboratories Incorpor...
Broadway & Seymour Inc
HotData Inc.
Ponti Restaurant
Rubios Fish Tacos
Jolly Ranch

Business Name: Rogue Wave Software
Address: 260 SW Madison Av
City: Corvallis
State: OR Zip Code: 97333
Phone Number: (541)754-3010 Fax: (541)757-6650
E-mail: President@RogueWave.com
URL: www.roguewave.com

Personal Contacts | Business Information | Geography | Notes

No. Employees: 250
Year Established: 1989

Contact List

Business Name	Address	City	State	Zip Code
Rogue Wave Software	260 SW Madison Ave	Corvallis	OR	97333
Rogue Wave Software Techni...	260 SW Madison Ave	Corvallis	OR	97333

Retrieve Next | OK | Cancel

Rogue Wave Software

HotData, Inc.
Price: HDK is free to qualified developers; end-users of HotData-enabled applications pay a usage fee (usually between US$.02 to US$.15) for the data they acquire; discounts are given for large batch requests.
Phone: (800) HOT DATA or (512) 646-6000
Web Site: http://www.hotdata.com

Mustang Announces IMC Architect

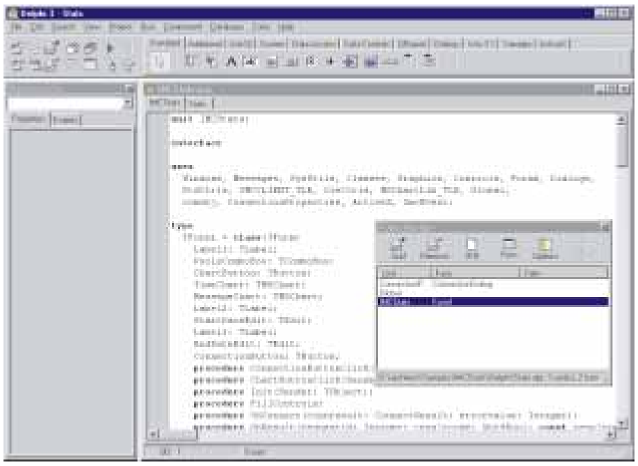
Mustang Software, Inc. announced the availability of ***IMC Architect*** for its Internet Message Center (IMC). By integrating IMC's client features into third-party desktop applications, call centers can use IMC Architect to reduce the number of desktop applications required to handle customer e-mail.

Application developers can use IMC Architect to add IMC real-time statistics to existing monitoring tools, or e-mail tracking features to existing enterprise applications, computer-telephony integration systems, or any proprietary enterprise application.

IMC Architect leverages Microsoft's COM technology to ensure interoperability in development environments, including Delphi, Visual J++, Visual Basic, and Perl.

This open, standards-based architecture provides flexibility to integrate new applications into call centers as business needs evolve.

Mustang Software, Inc.
Price: US$7,500 for an enterprise site license.
Phone: (805) 873-2500
Web Site: http://www.mustang.com



Excel Software Ships WinA&D 2.1

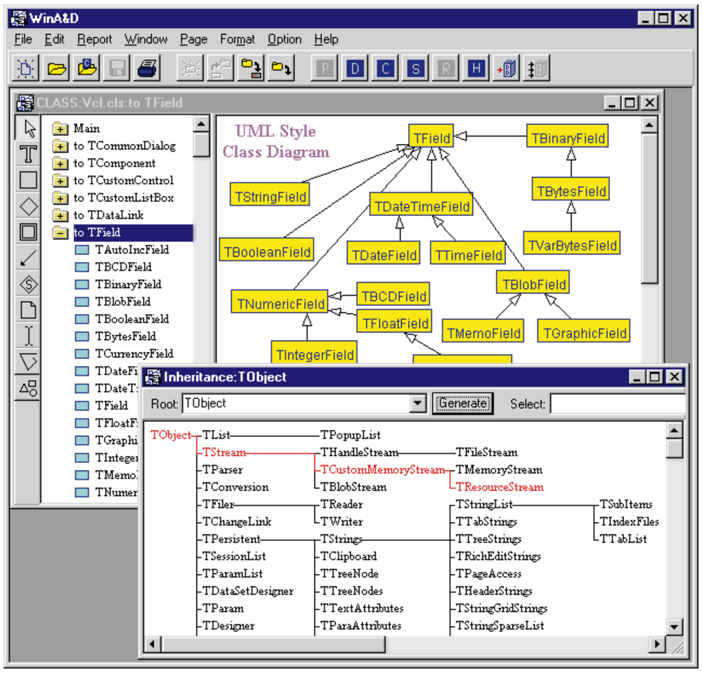
Excel Software announced the availability of ***WinA&D 2.1***, which supports system analysis, requirements specification, software design, and code generation for a range of software engineering methods and notations. Version 2.1 adds a Contents view for diagram editors, built-in zip archives for project documents, the WinA&D API, enhanced color support, and automated inheritance graphs for object-oriented designs. WinA&D 2.1 is available in Standard, Desktop, Educational, and Developer editions.

WinA&D 2.1 supports diagram editors for data, class, state, object, structure, and task models.

Instant inheritance graphs can be generated from root classes in the object-oriented design dictionary. Sophisticated algorithms provide a concise inheritance graph to express the class inheritance structure of design projects containing thousands of object classes. Designers can edit class properties from the inheritance graph.

COM automation is used to provide the WinA&D API. User-developed tools can supplement features in WinA&D, access document data, and be executed as stand-alone programs or as supplemental WinA&D commands. Languages such as Delphi, Visual Basic, Java, and C/C++ can access the WinA&D API.

Excel Software
Price: US$695, Standard; US$1,295, Desktop; US$845, Educational; US$1,995, Developer; upgrade pricing is available. WinA&D 2.1 can be purchased as a single-user, five-user, or unlimited-user site license.
Phone: (515) 752-5359
Web Site: http://www.excelsoftware.com



```
TObject┬TList──────────────TPopupList
       ├TStream────────────┬THandleStream────────TFileStream
       ├TParser            ├TCustomMemoryStream┬TMemoryStream
       ├TConversion        └TBlobStream        └TResourceStream
       ├TFiler─────────────┬TReader             ┌TStringList──────────┬TSubItems
       ├TChangeLink        └TWriter             ├TTabStrings          ├TIndexFiles
       ├TPersistent────────┬TStrings────────────├TTreeStrings         └TTabList
       ├TSessionList       ├TClipboard          ├TRichEditStrings
       ├TParamList         ├TTreeNode           ├TPageAccess
       ├TDataSetDesigner   ├TTreeNodes          ├THeaderStrings
       ├TParam             ├TTextAttributes     ├TStringGridStrings
       ├TDesigner          ├TParaAttributes     ├TStringSparseList
```

AnyWare Offers AppTools VCL 1.04

AnyWare Ltd. is offering ***AppTools VCL 1.04***, which contains four components and detailed example applications with source code for Delphi and C++Builder.

The *TWizard* component provides drop-in Wizard functionality. Developers can drop this component onto a form that has a *TNotebook* component and two *TButton* or *TBitBtn* components, set the corresponding properties of *TWizard*, and define the routes that can be taken using the custom property editor provided.

TSplashScreen offers flexible drop-in splash screen capability options that allow developers to customize the behavior of the splash screen.

TTipOfTheDay enables developers to create a drop-in Tip Of The Day form by dropping the component on a form, assigning some strings to the *Tips* property, and running the application. No code is needed for basic uses. *TTipOfTheDay* automatically stores the "last tip shown" information in an .INI file or the registry.

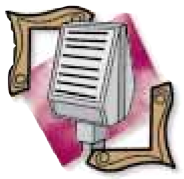
The *TBrowseButton* component creates a **Browse** button that allows the user to select a file using a standard Open dialog box, and assigns the selected file's name to an associated edit control.

AppTools VCL 1.04 is a native Delphi VCL and comes with a comprehensive Delphi- and C++Builder-compatible Help file.

AnyWare Ltd.
Price: US$20 (includes half-price discount on major upgrades and free minor version upgrades).
Fax: +44 0 117 973 6888
Web Site: http://www.anyware.co.uk

News LINE

October 1998



INPRISE Named Among Top Enterprise Vendors

Scotts Valley, CA — Software Magazine, a monthly publication for senior information technology managers and enterprise software professionals, has recognized INPRISE Corp. as a premier provider of enterprise-class application development tools in its annual "Software 500" listing of top corporate software vendors.

In addition to corporate recognition, INPRISE's Borland family of enterprise development tools has received a number of recent industry awards and positive product reviews. The company's Borland JBuilder was selected the industry's best Java development environment by *VARBusiness* and *BYTE* magazines, while JBuilder and Borland Delphi were named among the top 100 software products of the year by *Windows Magazine*.

Eshed Announces Gentia Wizard

Or Yehuda, Israel — Eshed Information Systems announced it has developed Gentia Wizard, a solution for generating reports for the Datawarehouse Gentia tool. The system works in the Windows 95 environment and was developed using Delphi 3.

The system uses a Borland Database Engine on an Oracle database and guides users through nine steps for report building. The user chooses the dimensions and the elements to be included in the report (including defining hierarchical dimensions and dimension nesting levels), creates integrated and computerized elements, and sets the calculation order and the order of appearance of all the elements in the report. At the end of the process, the system exports an interface file retrieved by the Gentia Wizard and presents the report as requested.

The dimensions and the elements presented by the system have a structure based on a hierarchical tree, offering the user rapid (and intuitive) access to the data.

Eshed Information Systems is involved in the development of information systems using Delphi and an Oracle database.

INPRISE and Referentia to Deliver JBuilder Training Software

Scotts Valley, CA and *Honolulu, HI* — INPRISE Corp. and Referentia Systems Inc. announced the availability of a computer-based multimedia training tool for software developers who want to learn advanced Java application development techniques. Referentia for JBuilder is a modular and integrated learning system for Borland JBuilder 2, INPRISE's family of visual development tools for building corporate and enterprise software applications with Java.

INPRISE also launched a JBuilder Web site to help developers build, manage, and deploy sophisticated Java applications using JBuilder 2's Java deployment server capabilities.

The first volume of the training system, "Volume I: Getting Productive in JBuilder 2," is available to JBuilder 2 users and other interested parties for US$9.95. Volume I offers a starter set of more than 15 lessons, overviews, and concept animations about working efficiently and productively in Borland JBuilder 2. Topics covered include the JBuilder 2 Editor, AppBrowser, BeansExpress, Debugger, and more.

The Volume I starter CD and information on additional volumes are available from Referentia Systems at (800) 569-6255 or (808) 396-1232, or online at http://www.referentia.com/jbuilder.

INPRISE Centralizes European Organization

INPRISE Corp. announced the centralization of its European operations into one region headquartered in Amsterdam, and named Gidi Schmidt Vice President and General Manager for the company's European operations.

The new structure is part of a global initiative to structure INPRISE's operations to reflect the company's focus on enterprise-level products and services. Under the European structure, sales, marketing, and administration will be coordinated from INPRISE's European headquarters in Amsterdam. Previously, INPRISE was organized into Western and Eastern European regions.

Errors and Omissions

Some major errors slipped by us in the September, 1998 issue of *Delphi Informant*. Both occur in Bill Todd's article, "Procedure Variables." On page 18, it's stated that method and function pointers are declared this way:

```
type
  TSomeProc = (SomeInt: Integer);
  TSomeFunc = (SomeString: string): Boolean;
```

This is incorrect; the **procedure** and **function** keywords are missing. Here are the correct declarations:

```
type
  TSomeProc = procedure(SomeInt: Integer);
  TSomeFunc = function(SomeString: string): Boolean;
```

A similar error appears on page 19 regarding method pointers. Here is the correct code:

```
type
  TSomeProc = procedure(SomeInt: Integer) of Object;
  TSomeFunc = function(SomeString: string): Boolean of Object;
```

Thanks to Jay Lazarus for the correction. We apologize for any confusion these errors caused.

By Dennis Butler

Crystal Reports

Interfacing with the Leading Report Writer

Crystal Reports is an impressive and powerful reporting tool that is underused in the Delphi community. From HTML exporting options to direct e-mail connectivity, Crystal is a time-tested, robust reporting tool that is connected with a simple-to-use and complete component for Delphi. This article will focus on obtaining the maximum benefit from Crystal, describing in detail the benefits and disadvantages of incorporating it into Delphi applications.

Because we're going to be looking at Crystal from a Delphi perspective, this article won't introduce all the capabilities of Seagate's report writer. Instead, it will review the methods required to obtain maximum flexibility from the product using Delphi and the Crystal environment together. This article assumes the reader is familiar with creating Crystal reports.

Some History

Although Crystal is touted as the "best-selling report tool," is bundled with major tools (e.g. Visual Basic), and has won many industry awards, Delphi developers have either a love or hate relationship with it. Some love its flexibility, readily available support, and wide range of features. Because it's widely used in the computing industry, and Seagate regularly develops new releases, using Crystal should be a safe way to ensure your applications can use newer technologies. Developers who loath the product, however, point out the historically poor Delphi support and the moderate footprint it leaves in the form of DLLs in your Windows system directory and elsewhere.

I've used the product enough to appreciate what it has to offer Delphi developers, however frustrating some things have been along the way. Moving forward, Seagate has made a concerted effort to reach the Delphi programming community with improved functionality and support.

Printing a Simple Report

The primary interface to Crystal from Delphi is the *TCrpe* component, which is available with the professional version of Crystal, and from the Seagate Web site (http://www.crystalinc.com). The Crystal component has evolved quite a bit since the original version for Delphi 1, so developers who haven't looked at the capabilities since then will want to revisit it. Here's a simple introduction.

To print an existing report using the *TCrpe* component, drop the component onto your Delphi form and use code similar to:

```
procedure TfrmPrintRpt.btnPrintClick(Sender:
          TObject);
begin
  with Crystal1 do begin
    Destination := toPrinter;
    ReportName  :=
      'C:\CRYSTAL\DATA\SIMPLE.RPT';
    Execute;
  end;
end;
```

We've used the basic properties of the component to print the report. The *Destination* property contains a list of several possible destinations, which will be covered in detail later. The *ReportName* property is simply the physical location of the report. The *Execute* method prints the report.

Simple, isn't it? If only all applications were this simple. However, in projects of any merit, there will be quite a bit more code involved than this; often the design requires complicated intervention to produce the desired results. Our next step in examining the component is to look at the *Selection*, *Formula*, and *Sort* properties of the component.

Selecting Reports to Print

In the previous example, the report and information in the tables linked to that report were printed. To limit information output by the component, we use the *GroupSelectionFormula* and *SelectionFormula* properties. These properties are similar to the *TStringList* type and hold the criteria by which the report will be limited. The *SelectionFormula* limits what is output on an individual record level, while the *GroupSelectionFormula* limits what is output for an entire group band in the report. Let's look at an example:

```
with Crystal1 do begin
  // Always clear SelectionFormula before every print.
  SelectionFormula.Clear;
  // Select active customers.
  SelectionFormula.Add('{ Customer.Status } = "Active"');
  // Select date range.
  SelectionFormula.Add(
    ' and { Sales.Sale_Date } > Date(1997,12,31)');
  GroupSelectionFormula.Clear;
  // Determine groups to show.
  GroupSelectionFormula.Add('Sum({ Sales.Amount },
    { Customer.Cust_Id }) >= 20000');
  ReportName := 'C:\CRYSTAL\DATA\20KSALES.RPT';
  Destination := toWindow;
  Execute;
end;
```

This report has a simple master-detail relationship between the Customer and Sales tables and has a group band on CustId. The information shown in the report is a list of sales for a particular customer. The *SelectionFormula* code is limiting the output to active customers and sales that have occurred in 1998. The *GroupSelectionFormula* is limiting the output to show only customers who have total sales for 1998 greater than $20,000. Setting the *Destination* property to *toWindow* shows the report in a preview mode.

Now we're getting somewhere. In this example, it's easy to see how the report can be manipulated using the *TCrpe* component. If we wanted, we could feed information in from Delphi controls. However, there are important considerations to be made when using the *SelectionFormula* property to select records.

When a Crystal report is run, a temporary table is created by the report engine that is a composite table of all the tables in the report. *SelectionFormula* limits the size of the Crystal Report table that is created, and selects records from the table to display. While a report is being generated, a dialog box will appear on which you will see a status bar reading something like the following:

of #### ##%

This is telling us the number of records that Crystal has selected, how many records total have been processed into the temporary table, and what percentage of the process is complete. Based on what fields are selected for use in the selection formula, there can be massive performance differences, even with the same output. Let's look at a two-table example, where a Customer table has 800 records, and a Sales table has 800,000 records (1,000 records for each master record).

With these two tables linked correctly in Crystal (link coming from Customer pointing to Sales), and having printed the entire list of customers and sales, we will yield an 800,000-record temporary Crystal table that will take a few minutes to generate. To see only one master record and its 1,000 detail records, we would set the selection formula to `{Customer.CustId}=#`. This will result in a temporary table of 1,000 records that is created almost instantaneously. If we were to set `{Sales.CustId}=#`, it would process all 800,000 records, even though the output would again be only the one master record and its 1,000 detail records. It's important to realize when running the reports that, whenever possible, the primary linking and selection of records should be from the master table.

Every time we run a report, we must remember that the *SelectionFormula* may hold information from the last time a report was run, so it's safest to clear it before re-assigning values to prevent the report from giving incorrect results. Note that creating more than one *SelectionFormula* only requires that some operator be used to connect them (i.e. and, or). When this formula is evaluated by Crystal, it's taken as an entire expression, so the following code fragments are identical:

```
// Example 1.
with Crystal1.SelectionFormula do begin
  Clear;
  Add('{ Customer.Status } = "Active"');
  Add(' and { Sales.Sale_Date } > Date(1997,12,31)');
end;

// Example 2.
with Crystal1.SelectionFormula do begin
  Clear;
  Add('{ Customer.Status } = "Active" and ' +
      '{ Sales.Sale_Date } > Date(1997,12,31)');
end;
```

One other item to note about using the *SelectionFormula* property is that it doesn't replace a selection formula set from within the report itself. Items added through the *TCrpe* component work in addition to any selection criteria already set for a record, or a group in the report through the Crystal environment.

Formula and Sorting Options

The two other main properties for the *TCrpe* component that affect the formatting of the report are the *Formulas* and *SortFields* properties. The *Formulas* property lets the programmer change the calculation or value of any existing formula field within the specified report. The *SortFields* property lets the programmer sort any or all tables in the report in ascending or descending order. (Note: Both properties are declared as arrays of strings, not as a *TStringList*.) Let's look at an example:

```
with Crystal1 do begin
  SelectionFormula.Clear;
  // Clear the list by setting [0] item to blank.
  SortFields[0] := '';
  // Sort on CustName in descending order.
  SortFields[0] := '-{ Customer.CustName }';
  // Then sort sales fields in ascending order.
  SortFields[1] := '+{ Sales.Amount }';
  // Clear any old formulas.
  Formulas[0] := '';
  // Set report title to user-defined string from
  // edit box edtTitle.
  Formulas[0] := 'fmlaTitle="'+edtTitle.Text+'"';
  ReportName := 'C:\CRYSTAL\DATA\SORTSALE.RPT';
  Destination := toWindow;
  Execute;
end;
```

As with the *SelectionFormula* and *GroupSelectionFormula* properties, the *SortFields* and *Formulas* properties have equivalents in the report itself. In this example, the report has a formula in it named *fmlaTitle*. By setting the formula through code, we change whatever it's set to in the report to our own value based on a value from an edit box. The *SortFields* property simply lists the table and field by which to sort, and prefixes it with the plus (+) or minus (-) characters to determine sort direction. Setting the value of the [0] array item of either of these properties, it clears all previous values entered.

In our simple example, we only put a string value in our Crystal formula field. Crystal has its own programming language with a wide variety of functionality. That topic is outside the scope of this article, but programmers should generally take advantage of the built-in functions that Crystal provides for formula fields.

Destination

The *Destination* property is where the real benefits of using Crystal are realized. By setting this to a valid value, we can send the report to a variety of outputs. We have already seen in the previous examples how to send reports to the printer and to the screen for previewing. The following examples show all that can be accomplished.

Example 1: Export the report to a file. This requires a few additional properties to be set, as shown in the following code. Although it's for Microsoft Excel 5.0, it works for all file outputs, as long as the *PrintFileType* property is set to the appropriate type:

```
with Crystal1 do begin
  Destination   := toFile;
  PrintFileType := Excel5;
  PrintFileName := 'C:\TEMP\MASTER1.XLS';
  ReportName    := 'C:\CRYSTAL\DATA\REPORT1.RPT';
  Execute;
end;
```

Setting the destination of the component to *toFile* requires the *PrintFileType* property to have a value specifying what format the report will be exported to. *PrintFileType* contains many useful export options for the report, including HTML, Word for Windows, several text formats, Excel or Lotus 123, and many others.

This is a simple example of how easy it is to use the full power of Crystal. Instead of having options limited to output to screen, or output to printer (as with other report writers), Crystal opens the door for user- or programmer-defined, ad hoc reporting. For example, the user can print reports to Excel where the data can be manipulated further. Given the market share that Excel has in today's computing industry, this is an important capability.

Example 2: Send the report to e-mail via MAPI (Microsoft Mail or cc:Mail v8) as an attachment in HTML format. For example:

```
with Crystal1 do begin
  Destination   := toEmailViaMAPI;
  PrintFileType := HTMLNetscape;
  EmailToList   := 'Dennis Butler';
  EmailSubject  := 'HTML Document';
  EmailMessage  := 'Attached is the HTML Document.';
  ReportName    := 'C:\CRYSTAL\DATA\REPORT1.RPT';
  Execute;
end;
```

The only thing this requires is that a MAPI-compliant mail service be set up for the target machine, and that the *EmailToList* property contains a valid address from your mail address book.

DataFiles and the *OnLoadDataFiles* Event

Crystal lets you link tables in a variety of ways in the report environment, including specifying which direction to link the tables, and in what order to load information if there's more than one detail table. Unfortunately, linking to two detail tables, or trying to set multiple conditions on linked tables, can lead to trouble in Crystal. For example, if you have a master table linked to two detail tables, one having two matching records and one having three matching records, depending on how the detail band is arranged, your output will sometimes show three records for each detail table. Other times, information for some tables can be removed entirely based on selection criteria set for different tables. Under certain conditions, Crystal will duplicate or exclude information in the report in a multiple-detail table situation.

The Open Sales and Send Schedule report is an example (see Figure 1). This customer has ordered three items, two of which are closed. This customer also has two scheduled shipments. The desired report output should print all non-closed items from the Sales table, and the overall schedule. To do this, I created a Crystal report that links CUSTOMER to ORDERED and SCHEDULE, with a selection formula set to exclude any records from the

ORDERED table that have a closed status. The linking from Crystal is shown in Figure 2.

The report has one Customer group band, with a detail band that contains two fields: the Amount field from the ORDERED table, and the Amount field from the SCHEDULE table. The expected output would be one item in the ORDERED QTY column of the report, and two items in the SENT QTY column of the report. Instead, we get the output shown in Figure 3.

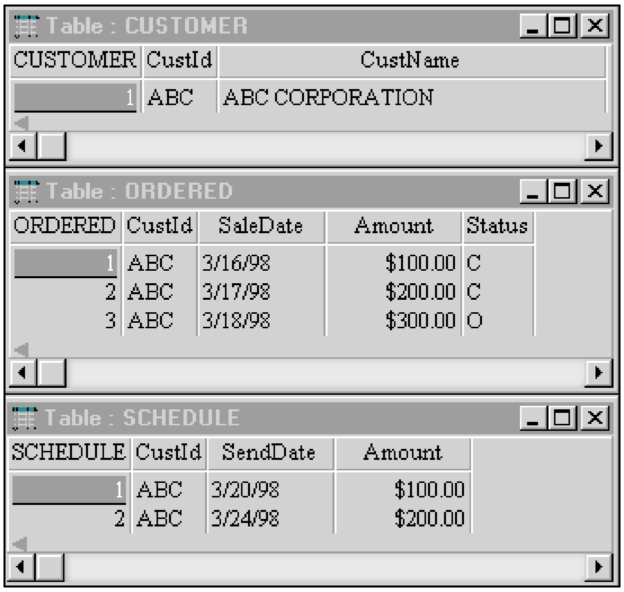


Table : CUSTOMER

CUSTOMER	CustId	CustName
1	ABC	ABC CORPORATION

Table : ORDERED

ORDERED	CustId	SaleDate	Amount	Status
1	ABC	3/16/98	$100.00	C
2	ABC	3/17/98	$200.00	C
3	ABC	3/18/98	$300.00	O

Table : SCHEDULE

SCHEDULE	CustId	SendDate	Amount
1	ABC	3/20/98	$100.00
2	ABC	3/24/98	$200.00

Figure 1: Table contents for sample report.

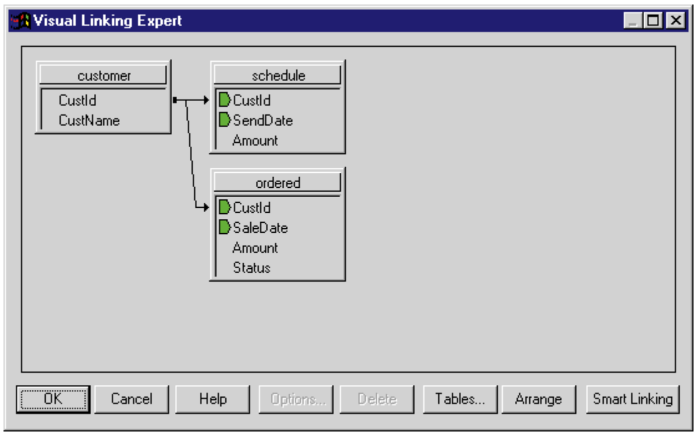


Figure 2: Link configuration from within Crystal Reports.

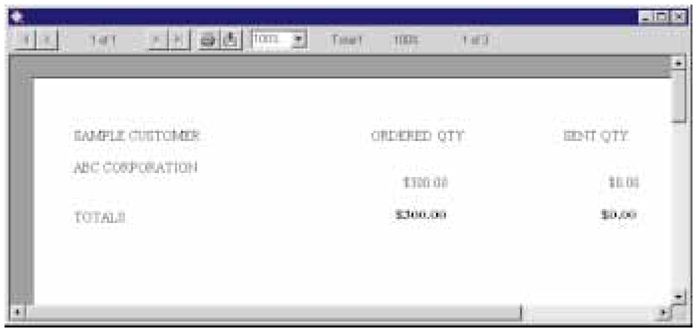

```
SAMPLE CUSTOMER          ORDERED QTY      SENT QTY
ABC CORPORATION
                            $300.00          $0.00
TOTALS                      $300.00          $0.00
```

Figure 3: Sample report output showing incorrect totals.

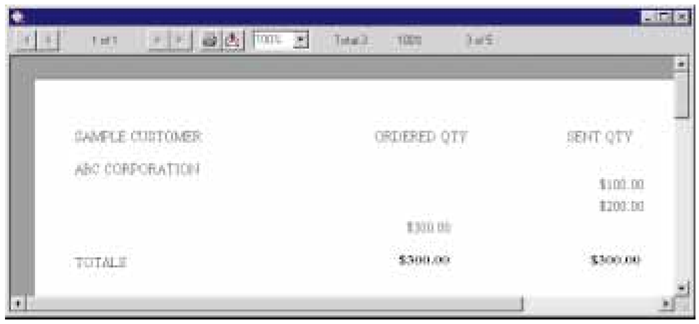

```
SAMPLE CUSTOMER          ORDERED QTY      SENT QTY
ABC CORPORATION
                                           $100.00
                                           $200.00
                            $300.00
TOTALS                      $300.00        $300.00
```

Figure 4: Sample report output showing correct totals.

As you can see, we are completely missing the SENT QTY information, even though there are two items in the database. This kind of mistake can be hard to spot, especially if there is one master with multiple-detail tables and selection restrictions in place. This problem is a result of the nature of the composite temporary table that Crystal creates for each report. The selection formula took out the two ORDERED QTY records that were closed, but those two records held the information for the two missing SENT QTY items. As a result, the output is missing this information, although we didn't want to exclude those items from the report.

One solution is to select the option **Look up all of one, then all of the other** from the **Options** button under the Visual Linking Expert of Crystal Reports. Unfortunately, the output for a report with multiple-detail tables will appear as shown in Figure 4.

Here, we have the correct output, but it doesn't look entirely professional, because the detail items do not share lines. In this situation, there are only a few items, so it may not matter. But in reports with large amounts of data, it can appear awkward when information from different columns shows up on new lines of the report. When the limitations of linking tables in Crystal become evident, we can choose another option: create the answer table ourselves. In a single-user environment, this table can be linked directly into the report and created each time the report is run from Delphi. However, in a multiuser environment, this option may not be practical because multiple users may try to recreate the temporary table at the same time.

One way to solve this is to switch the name or location of certain tables at run time through the *OnLoadDataFiles* event. In the following example, the table created in Delphi is named as the person's login ID. This temporary table is then connected to Crystal through the *OnLoadDataFiles* event. We cycle through the *DataFiles* property to look for the appropriate table and swap it. The structure of the temporary table to which the report points in design mode and the structure of the table, which is created through Delphi, must be identical.

The two important parameters are *Count* and *Cancel*. *Count* is the number of databases in the report, and

Cancel tells whether to cancel the report. This is useful if this event cannot find the temporary table to which it is trying to link:

```
procedure TfrmPrintRpt.Crystal1LoadDataFiles(
  Sender: TObject; const Count: Integer;
  var Cancel: Boolean);
var
  iCtr : Integer;
  sFileName : TFileName;
begin
  // Set file name to user id. sDataPath is set by my
  // application to be the location of the data.
  sFileName := sDataPath + 'TMP' +
    tblLogin.FieldByName('UserId').AsString + '.DB';

  // Find the appropriate table and swap it.
  for iCtr:=0 to Count-1 do
    if ExtractFileName(
        Crystal1.Datafiles[iCtr]) = 'SALES.DB' then
      Crystal1.DataFiles[iCtr] := sFileName;

end;
```

This method is useful if what you are trying to accomplish is beyond what Crystal allows you to do by default. This ability allows practically any manipulation of data to be sent to a report in the desired format without trying to get the tables to link correctly from within Crystal.

Miscellaneous Tricks and Tips: Installation and Upgrading

With Crystal Reports 6.0, and the new version of the component available to Delphi developers, we can be confident that things will run as expected. I had the unfortunate experience of trying to get Crystal Reports version 4.5 (32-bit) to work with Delphi through the old version of the *TCrpe* component. After countless hours with Seagate technical support, multiple patches from Seagate, and increasingly frustrated users who simply wanted 32-bit reporting, I was very close to abandoning Crystal for a more programmer-friendly product.

Versions 5.0 and 6.0 of Crystal, as well as the new version of the component, have proven to be a lot more reliable and worthwhile. If you must venture into the pre-5.0 version world of Crystal, however, be aware of a few things:

- Version 4.5 (16- and 32-bit) requires patches from Seagate to work. The DLLs from the installation disks have memory problems and will cause GPFs or Access Violations on your machine.
- Version 4.5 (32-bit) initially had no Paradox table support when it was released. You can contact Seagate for the missing DLL (P2BBDE.DLL), which you should put into your Windows \System folder.
- Version 4.5 or earlier will cause severe Access Violations upon exiting your Delphi applications if you are using BDE version 3.5 or later.

The good news for programmers who are planning to upgrade from an earlier version of Crystal is that the reports themselves are completely backward-compatible, so version 4.0 reports will run in the 6.0 environment. However, it's good practice to print all reports that have been upgraded. I've found that some formatting is lost when printing old reports in the new environment — for example, previously underlined fields were no longer underlined, and some amount fields that were set to hide commas in the old version show commas in the new version. It doesn't seem to happen in all circumstances, so it's good practice to fully test the converted reports.

Divider Lines

One interesting trick I ran into evolved from a client request regarding divider lines in a report. The client wanted to be able to selectively turn on/off divider lines in the detail sections of certain reports. The easy way out would have been to create two reports — one with divider lines and one without — but I didn't want to have to maintain two reports. Instead, I found an alternative solution.

In the detail section, create a regular line that doesn't have any other items on the same report line. Create a formula field named *fmlaDivider* without a value for its formula, and place the field on the same line as the divider line we've drawn. Shrink the size of the field to be as small as possible and right-justify it. Next, make sure that the group section containing the divider line has the setting **Suppress Blank Lines** checked. This will take out the blank spaces normally left by null fields.

From Delphi, pass a value to this formula based on the user selection. If the user does *not* want a line, pass an empty string. If the user *does* want a line, pass a single character padded with spaces to the left of the character. If the formula field is blank when the report is run, it will "suppress" the line or page number on the same line as the null field and it won't appear. For example:

```
Formulas[0] := '';
// If checkbox is checked, then show divider line.
if chkUseDivider.Checked then
  // fmlaDivider is the formula name in the report itself.
  Formulas[0] := 'fmlaDivider="              X"';
else
  Formulas[0] := 'fmlaDivider=""';
Execute;
```

Because the formula was left-justified when the value was passed to the report, we won't see it in the output, but the line will appear. The line will disappear when the formula has passed an empty value because of the **Suppress Blank Lines** option.

Crosstab Reports

There are many situations where it makes sense to use the Crystal Reports Crosstab report. It's a non-standard report that shows the value of one field as the rows, the value of another field as the columns, and the resulting field — a calculation of a field(s). The format shows data in an alternative way.

However, the Crosstab report from Crystal is very restrictive. It allows very limited access to the fields in the crosstab itself, and sorts the columns and rows in the default order for the selected field. Fields can't be added or removed from the crosstab layout, except from the Crosstab Setup itself. I ran into a problem creating a report that needed to have the column fields put in a different order than the Crystal alphabetical default. I wanted to display the value of one field, but sort the columns by another. To do this, I again used the ability to hide certain elements of a field.

The first step is to create a temporary table identical to the crosstab table in the report, except the field used for the column should be comprised of several fields. This field was made of a string concatenation of the sorting field, a large number of spaces, and the field to display. By using this temporary table in the report and right-justifying the field in the crosstab, the sorting field isn't visible because the field wasn't stretched far enough to show it.

For example, we have a table used by a grocery store for inventory that contains three fields — Category, Date, and Amount — from which to create a crosstab report. Some of the categories for this table are Vegetables, Canned Goods, Deli, and Anything Else. In a Crystal crosstab with the Category set as the column, Date set as the row, and the Amount field as summary, the result will be as follows:

	Anything Else	Canned Goods	Deli	Vegetables
1/1/97	$200	$100	$80	$50
2/1/97	...			

This is fine unless the user wants to re-arrange the order of the columns. Crystal sets an alphabetical order for the columns. To overcome this, we must create a temporary table. The only thing we need to do is change the Category items so they are prefixed by particular letters. For example:

"A	Vegetables"
"B	Canned Goods"
"C	Deli"
"D	Anything Else"

Now our crosstab will look like this:

	Vegetables	Canned Goods	Deli	Anything Else
1/1/97	$50	$100	$80	$200
2/1/97	...			

Because we stretched the fields of the crosstab to fit the original captions, the prefix letters won't be viewed on the final report.

Changing Printers

In Crystal, there are three properties available to change the printer. However, there is an additional public variable that also needs to be modified. The *PrinterName*, *PrinterDriver*, and *PrinterPort* properties need to be changed, as well as the public variable *PrinterMode*. Because *PrinterMode* doesn't appear in the Object Inspector, it can be frustrating to try to change the printer at design time.

Sending Reports via VIM

With versions of Crystal later than version 5.0, support for VIM has been removed. However, Lotus released cc:Mail v.8 in 1997, which was MAPI-compliant, so mail is still possible through cc:Mail. That said, the *TCrpe* component still has a setting for the *Destination* to be *toEmailViaVIM*. Using this option assumes you are using an earlier version of Crystal, and will produce a DLL version error if you try to use it with Crystal Reports 5.0 or later.

Seagate has also removed export support for WordPerfect, Word for DOS, and Quattro Pro. These are important considerations to be made if you are upgrading mail systems or upgrading an existing application that uses these options.

Reducing the Run-time Footprint

Distribution of Crystal leaves a sizable amount of files on the target machine. The latest CRPE32.DLL report engine itself weighs in at a hefty 3.5MB. However, some of this overhead can be reduced by taking a few steps. You can use the Report Distribution Expert within Crystal to see all necessary DLLs for any specific report. This isn't always the best way to determine which DLLs are necessary, and you may need to investigate further. Crystal segments export DLLs and function calls from the report into different DLLs, so if a new version of a report with new functionality was sent to a target machine, it may look for a DLL that wasn't originally installed.

The best way to approach which DLLs to select is to include all of them, then decide which ones will never be used. Following are the 32-bit DLLs distributed with Crystal:

- P2*.DLL — Database DLL, depends on table format used. Usually only one is necessary.
- PG32.DLL — Required for any graphing in reports.

Destination options:

- U2DDISK.DLL — Disk file destination
- U2DMAPI.DLL — MAPI format (Microsoft Mail, Microsoft Exchange, cc:Mail v8)
- U2DPOST.DLL — Microsoft Exchange Public Folders (older version of Crystal)
- U2DVIM.DLL — VIM format (cc:Mail, Lotus Notes, WPOffice, etc.)
- U2DNOTES.DLL — Lotus Notes

Export options:

- U2FCR.DLL — Crystal Reports, 16-bit format
- U2FDIF.DLL — DIF format
- U2FHTML.DLL — HTML format
- U2FODBC.DLL — ODBC data source
- U2FREC.DLL — Record format
- U2FRTF.DLL — Rich Text Format
- U2FSEPV.DLL — Comma Separated Values format

- U2FTEXT.DLL — Text format
- U2FWKS.DLL — Lotus 123 format
- U2FWORDW.DLL — Microsoft Word for Windows format
- U2FXLS.DLL — Microsoft Excel format (older version of Crystal)
- U2FDOC.DLL — Word for DOS and WordPerfect format
- U2FQP.DLL — Quattro Pro

Any of these DLLs can be excluded if they aren't used in an application. DLLs starting with "U2L" are called User Function Libraries. These DLLs are used for Crystal formulas. In addition to the user function libraries that come standard with Crystal, there are many additional DLLs available from Seagate's Web site. You can add quite a bit of new functionality to your reports by installing them. Programmers can also create their own DLLs to add new functionality to their reports. This extensibility of Crystal is very useful when creating new functionality that can be shared among many different reports.

Conclusion

Overall, Crystal Reports 6.0 gives developers a wide range of options and functionality. With its large user base, Crystal is constantly being improved and patches or updates are, for the most part, available in a timely manner. There is a large technical support staff at Seagate, with representatives who know Delphi and who use the *TCrpe* component, so it's much easier to get useful information quickly than in previous years when no Delphi support was available.

With its separate report development environment, creating reports doesn't have to be done by the Delphi programmer alone, but instead can be opened to anyone who is familiar with Crystal. Its wide array of export functionality provides a variety of useful ways to format and output your reports. The *TCrpe* component provides all the functionality we need, and there is rarely a situation when we need to delve into the Crystal API.

Although Crystal can sometimes be frustrating when it behaves in what appears to be an illogical manner, patience and workarounds make using it a worthwhile effort, given all the product's strengths. Although past versions were only partially functional with Delphi and created many problems, Seagate seems to have made a concerted effort to improve both the support and technical aspects of using Crystal Reports with Delphi. As the product continues forward, we can expect to see an even wider range of possibilities for programmers. ∆

Dennis Butler is a Senior Application Developer for Apogee Information Systems, an INPRISE Premier Partner specializing in multi-tier Delphi and JBuilder solutions. Mr Butler delivered several presentations at the original Borland Canada Conference in 1997. He has been designing and developing Delphi applications since the first release of the product.

By Neville Kelly

Reports in DLLs

Report Distribution Made Simple

The largest class of programs developed using Delphi are data programs, i.e. programs that maintain data stored in a database and extract information from the database by means of reports. Usually, the data maintenance part of these programs is relatively stable, but developers frequently need to add new reports or change existing reports to meet client requirements. If the program is developed as a single executable file, then the entire executable file must be distributed to clients to provide them with the new or changed reports. As a result, the file distributed is larger than necessary.

This article describes a technique to avoid the distribution problem. The reporting part of the program is distributed as a dynamic link library (DLL). Thus, only the DLL has to be distributed when reporting requirements change. The technique involves:

- report and parameter definitions stored in database tables,
- a report launcher form that is part of the main executable file,
- the report DLL containing the reporting code, and
- a separate report development and testing application.

Additionally, we'll describe simple implementations of two reporting "bells and whistles:"

- providing users with an online sample for each report, and
- implementing user report access control.

This article covers this technique as it relates to QuickReport, simply because all Delphi users have a copy of it. However, I first developed this technique for use with ReportPrinter Pro.

Report and Parameter Definitions

When designing the report and parameter data structures, we need to provide a means of displaying to users available reports and their parameters in a manner similar to a reporting menu. Also, we must provide appropriate prompts, hints, and parameter value control for each report and parameter. To store the report and parameter definitions, we could use .INI files, the registry, or a structured storage file. But, as we are dealing with a database program and have already incurred the overhead of a database management system, it's natural to use database tables.

The report and parameter table definitions are shown in Figure 1. RptId is the primary index of the report table, and the ParmRptId and ParmSeqNbr columns form the parameter table primary index. By linking RptId to ParmRptId, we can create a master-detail relationship where a report may have many parameters that will be in ParmSeqNbr order.

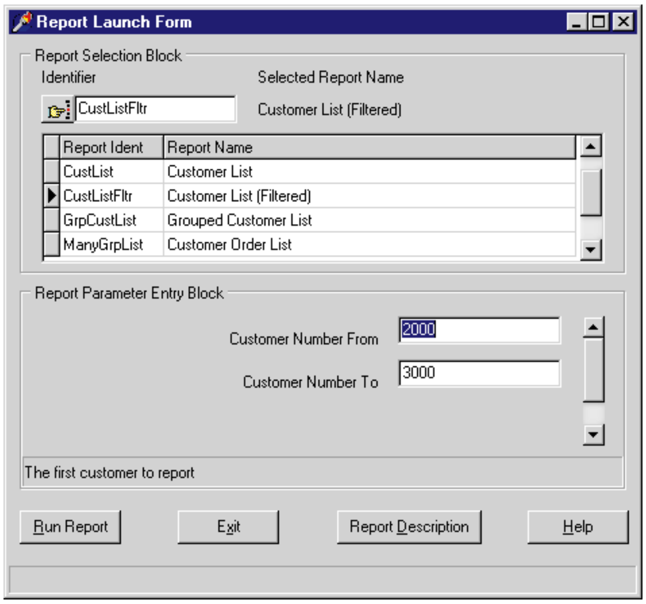
Report Launch Form

The Report Launch Form described in this article is a stand-alone program (ReportsTest.exe). However, in practice, it would be part of the main data maintenance program, and would be shown when the user selected a reporting option from the main menu. The form enables the user to select a report, enter parameters for the report, and run it. It provides the link between the data

Report Table				
Field Name	**Description**	**Data Type**	**Required**	**Usage**
RptId	Report Identifier	Alpha	Yes	Short name used to identify the report
RptName	Report Name	Alpha	Yes	Long descriptive name for the report
EdtRptCtxt	Help Context	Long	No	Help topic context identifier for the report
RptAccessLevel	Report Access Level	Long	No	A number appropriate to the report access level (the user must have an access level equal to or higher to run the report)

Parameter Table				
Column	**Description**	**Data Type**	**Required**	**Purpose**
ParmRptId	Report Identifier	Alpha	Yes	Links a set of parameter records to a report
ParmSeqNbr	Parameter Sequence Number	Short	Yes	Controls the order in which the parameters are displayed
ParmName	Parameter Name (Developer)	Alpha	Yes	Parameter name used to insert and extract a parameter value from the parameter list
ParmText	Parameter Name (User)	Alpha	Yes	Parameter name displayed to the user
ParmValue	Parameter Value	Alpha	No	Stores the parameter value
ParmMask	Parameter Mask	Alpha	No	Provides a standard Delphi field edit mask for the parameter at run time
ParmReqd	Parameter Required	Logical	No	Controls whether the parameter is required or optional at run time
ParmHint	Parameter Hint	Alpha	No	Provides hint to advise the user about parameter values

Figure 1: The report and parameter table definitions.


Report Launch Form

Report Selection Block

Identifier: CustListFltr

Selected Report Name: Customer List (Filtered)

Report Ident	Report Name
CustList	Customer List
CustListFltr	Customer List (Filtered)
GrpCustList	Grouped Customer List
ManyGrpList	Customer Order List

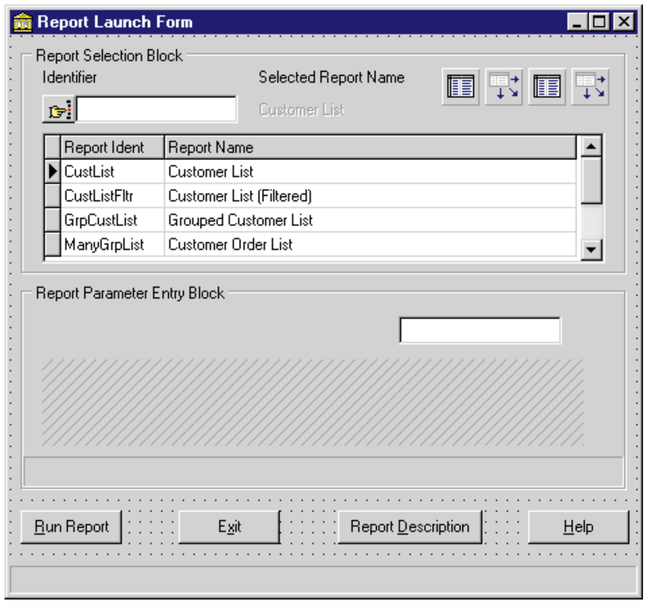
Report Parameter Entry Block

Customer Number From: 2000

Customer Number To: 3000

The first customer to report

Figure 2: The Report Launch Form at run time.


Report Launch Form

Report Selection Block

Identifier

Selected Report Name: Customer List

Report Ident	Report Name
CustList	Customer List
CustListFltr	Customer List (Filtered)
GrpCustList	Grouped Customer List
ManyGrpList	Customer Order List

Report Parameter Entry Block

Figure 3: The Report Launch Form at design time.

maintenance program and the report DLL. Figure 2 shows the form at run time, and Figure 3 shows the form at design time. The data access components on the form are:

- a Table component, *tblReport*, set to the report table;
- another Table, *tblParm*, set to the parameter table; and
- two DataSource components, *dtsReport* and *dtsParm*, connected to *tblReport* and *tblParm*, respectively.

The tables are connected in the desired master-detail arrangement by setting *tblParm*'s *MasterSource* property to *tblReport*, and its *MasterField* property to *RptId*. Additionally, we create persistent fields for all columns of both tables. In the **Report Selection Block**, there are four main controls:

- an Edit component, *edtRptId*, enabling the user to locate a report;
- a SpeedButton component, *btnEdtRptId*, located adjacent to *edtRptId*, enabling the user to start a new report locate process;
- a DBGrid, *grdReport*, showing the available reports; and
- a DBText, *lblRptName*, showing the current/selected report.

To set up the block's data controls, we set both *grdReport*'s and *lblRptName*'s *DataSource* property to *dtsReport*, and *lblRptName*'s *DataField* property to *RptName*. The *edtRptId.Text* property and the cursor for *tblReport* data set are linked by code in the *edtRptIdChange* procedure:

```
tblReport.Locate('RptId', edtRptid.Text,
                 [loPartialKey, loCaseInsensitive]);
```

This has the effect, as the user types into *edtRptId*, of moving the data set cursor (and hence the grid row pointer) to the record most closely matching *edtRptId.Text*. Additionally, we "auto fill" the *edtRptId.Text* property with the value of the current *tblReportRptId* field by using code in the *dtsReportDataChange* procedure:

```
edtRptId.Text := tblReportRptId.Value;
```

Thus, *grdReport*'s row pointer indicates the current report, *edtRptId* shows the current report identifier, and *lblRptName* shows its name ("grayed-out" to indicate that it is not selected). Users can type a new report identifier at any time by pressing the speed button, *btnEdtRptId*. The button's *OnClick* event simply clears *edtRptId.Text* and resets the state of those controls that may have been set previously.

If we create a Help file with a topic for each report, we can provide the user with an online sample of the current report. The sample Help topics in the file I've provided with the demonstration program (see end of article for download details) are quite basic and could, for example, be expanded by:

- giving detailed explanations of the effect of particular parameter values, and
- describing when (and when not) to run a report.

We associate the current report with the appropriate Help context identifier, and thus display the correct Help topic with code in the *btnRptDescClick* procedure:

```
Application.HelpContext(tblReportRptHelpCtxt.Value);
```

The user may select the current report by pressing Enter↵ while either *edtRptId* or *grdReport* has input focus. We do this by pointing *grdReport*'s *OnKeyDown* event handler to *edtRptId*'s *OnKeyDown* event procedure. The code is shown in the *edtRptIdKeyDown* procedure.

If the key triggering the event is Enter↵, this procedure does several things:

- *lblRptName* is enabled to indicate the report has been selected.
- UserAccessLevel is checked to see if it's sufficient to run the report. If it's not, the **Run Report** button is disabled and an error dialog box is displayed. Normally, the UserAccessLevel would be associated with the user at login, as set by the system administrator; however, in this demonstration program, the default UserAccessLevel is 5. You can override this by passing a parameter at run time (within Delphi, **Run | Parameters**).
- If the report has parameters, we activate controls in the **Report Parameter Entry Block**. Specifically, we: 1) display the hint for the first parameter, 2) show the parameter grid *grdParm*, and 3) move focus to *edtParmValue*.

If the report has no parameters, we simply move focus to the **Run Report** button.

The main controls in the **Report Parameter Entry Block** are:

- a DBControlGrid component, *grdParm*, connected to *dtsParm*; and
- a Panel, *pnlParmHint*, used to display *ParmHint* contents.

On *grdParm*, we have:

- a DBEdit, *edtParmValue*, connected to *ParmValue*; and
- a DBText, *lblParmText*, connected to *ParmText*.

To control parameter values, we may wish to alter the *tblParmParmValue.EditMask* and *tblParmParmValue.Required* properties at run time. By providing an EditMask, we ensure the user enters only valid report parameter values. By controlling the *Required* property, we ensure all essential parameters are entered, while still enabling optional parameters to be left blank. The following *tblParmAfterScroll* procedure code sets these properties for us:

```
tblParmParmValue.EditMask := tblParmParmMask.Value;
tblParmParmValue.Required := tblParmParmReqd.Value;
```

When the user advances from one parameter to the next, we need to invoke checking for the *Required* property. We cannot use *EDBEngineError* to process violation of the *Required* property because the field is defined as "not required" in the database definition (this allows us to have optional parameters). Instead, we must use the more general *EDatabaseError*. Where the user attempts to violate the *Required* property, we can customize the error message by checking the default error text and providing our own error text. The default error text is used for other *EDatabaseErrors*. This is shown in the *ProcError* procedure, which may be called from *edtParmValueExit* or *grpParmKeyDown* procedures:

```
procedure TfmLauncher.ProcError(E: EDatabaseError);
begin
  if Pos('must have a value', E.Message) > 0 then
    MessageDlg('This parameter is required, ' +
      'you must enter a value', mtError, [mbOk,mbHelp], 10)
  else MessageDlg(E.Message, mtError, [mbOk], 0);
end;
```

The *ExecReport* procedure runs the report (see Figure 4). But before we call it, we must store the report parameter values in the *TStringList* (*ParmsList*) passed as a procedure parameter. We do this (see the *btnRunClick* procedure) by looping through the Parm table and adding a Name=Value pair to *ParmsList* for each record with:

```
ParmsList.Add(tblParmParmName.Value + '=' +
              tblParmParmValue.Value);
```

```
// Uses RptId to create the appropriate report form.
// Processes ParmsList to resolve parameter values and
// allocate them to the appropriate query or table
// parameters. Runs the report and frees report form
// resource.
procedure ExecReport(RptId: string;
  ParmsList: TStringList);
begin
  fmDriver := TfmDriver.Create(Application);
  try
    if RptId = 'CustList' then begin
      ListForm := TListForm.Create(Application);
      try
        // This report has no parameters.
        ListForm.QuickRep.Preview;
      finally
        ListForm.Free;
      end;
    end;
    if RptId = 'CustListFltr' then begin
      ListForm := TListForm.Create(Application);
      try
        // Use report parameters to build table filter
        // string.
        with ListForm.QuickRep do begin
          DataSet.Filtered := True;
          DataSet.Filter := 'Custno > ' +
            ParmsList.Values['CustNbrFrom'] +
            'and CustNo < ' +
            ParmsList.Values['CustNbrTo'];
          Preview;
        end;
      finally
        ListForm.Free;
      end;
    end;
    if RptId = 'GrpCustList' then begin
      GrpListForm := TGrpListForm.Create(Application);
      try
        // Use report parameter to build query parameter
        // value.
        with fmDriver.qryCustomer do begin
          Close;
          ParamByName('pCoChar').AsString :=
            ParmsList.Values['CustName'] + '%';
          Open;
        end;
        GrpListForm.QuickRep.Preview;
      finally
        GrpListForm.Free;
      end;
    end;
    if RptId = 'ManyGrpList' then begin
      ManyGrpForm := TManyGrpForm.Create(Application);
      try
        with fmDriver.RepQuery do begin
          Close;
          ParamByName('pCustNo').AsInteger :=
            StrToInt(ParmsList.Values['CustNo']);
          Open;
        end;
        ManyGrpForm.QuickRep.Preview;
      finally
        ManyGrpForm.Free;
      end;
    end;
  finally
    fmDriver.Free;
  end;
end;
```

Figure 4: The *ExecReport* procedure.

Thus, *ParmsList* for the *CustListFltr* report might contain:

```
CustNbrFrom=2000
CustNbrTo=3000
```

Note that *ExecReport* is contained in the DLL Reports.dll. Thus, it's defined as:

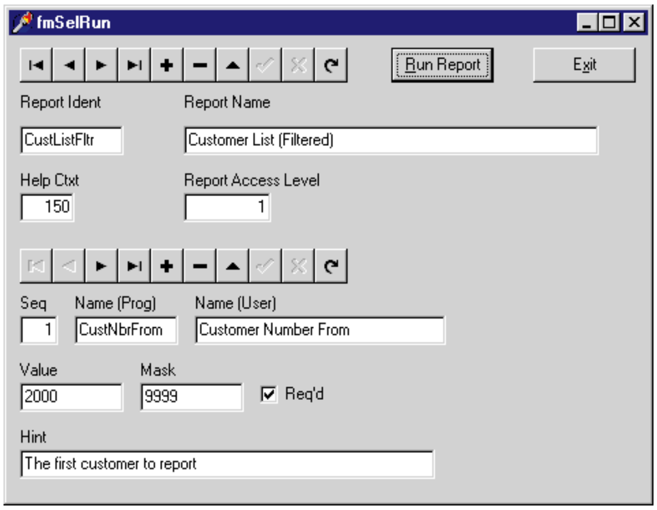
```
procedure ExecReport(Rptid: string;
  ParmsList: TStringList);
  external 'Reports' name 'ExecReport';
```

Report Development and Test Application

Next, we'll look at RptTestDev, the companion application used to develop and test reports. We need this application because we want to avoid the tedium of having to compile the DLL each time we make a change to one of our reports while we are in the development and test phase.

First, let's look at the user interface shown in Figure 5. The important thing to note about this unit is that the *TfmSelRun.btnRunClick* procedure code is the same as that for *TfmLauncher.btnRunClick*. That is, the ReportsTest and RptTestDev applications use the same code to run reports.

Next, we'll focus our attention on the Driver unit; it contains the definition for the *ExecReport* procedure. With *ExecReport*, we identify the report, resolve each parameter value, and allocate it to the appropriate report query or table parameter or use it to control report processing in some way. We then run the report. In this article, we've simply taken some of the reports supplied as demonstration reports with QuickReport and modified them slightly to illustrate various methods of using parameters with reports. Thus, the reports are quite simple, but are adequate to demonstrate the potential of the technique. Because we need to make it visible to the outside world,



Report Ident: CustListFltr
Report Name: Customer List (Filtered)
Help Ctxt: 150
Report Access Level: 1
Seq: 1
Name (Prog): CustNbrFrom
Name (User): Customer Number From
Value: 2000
Mask: 9999
[✓ Req'd]
Hint: The first customer to report

Figure 5: The user interface.

we add to the interface part of the Driver unit the **export** directive, and *ExecReport* is declared as:

```
procedure ExecReport(RptId: string;
  ParmsList: TStringList); export;
```

The basic structure of *ExecReport* is:

```
begin
  fmDriver := TfmDriver.Create(Application);
  try
    // A series of statements for each report to resolve
    // parameters and run the report.
  finally
    fmDriver.Free;
  end;
end;
```

Because we create the form *fmDriver*, we must protect it within a **try..finally** block so that the form resources are freed if a report fails in some way.

Next, we'll look at a simple report (*RptId* = *CustList*) with no parameters to reveal the code needed to run a report:

```
if RptId = 'CustList' then begin
  ListForm := TListForm.Create(Application);
  try
    // This report has no parameters.
    ListForm.QuickRep.Preview;
  finally
    ListForm.Free;
  end;
end;
```

As you can see, we simply create the report form, run the report, then free the form resources. Again, we need to ensure resources are freed by using a **try..finally** block. We can extend this into a slightly more complex report (*RptId* = *CustListFltr*) when we use the report parameters to build a string for the *Filter* property:

```
if RptId = 'CustListFltr' then begin
  ListForm := TListForm.Create(Application);
  try
    // Use report parameters to build table filter string.
    with ListForm.QuickRep do begin
      DataSet.Filtered := True;
      DataSet.Filter := 'Custno > ' +
        ParmsList.Values['CustNbrFrom']
        + 'and CustNo < ' + ParmsList.Values['CustNbrTo'];
      Preview;
    end;
  finally
    ListForm.Free;
  end;
end;
```

Note that *CustListFltr* actually uses the same report form as *CustList*; all we have done is set up a table filter. Note also that we get the value part of a Name=Value pair, i.e. we get the parameter value from the string list *ParmsList* with:

```
ParmsList.Values['Name']
```

For anything more sophisticated than a filter, we need to set the report table or query properties. Thus, we need to place the *QuickReport.DataSet* on the *fmDriver* form rather than on the report form. This is the case with the next report (*RptId* = *GrpCustList*). This report is a demonstration report, *GrpListForm*, with the Query component, *qryCustomer*, used as the *QuickReport.DataSet* instead of the Table. We set the *qryCustomer.SQL* property to:

```
SELECT * FROM Customer
WHERE Company LIKE :pCoChar
ORDER BY Company
```

and with the code:

```
if RptId = 'GrpCustList' then begin
  GrpListForm := TGrpListForm.Create(Application);
  try
    // Use report parameter to build query parameter value.
    with fmDriver.qryCustomer do begin
      Close;
      ParamByName('pCoChar').AsString :=
        ParmsList.Values['CustName'] + '%';
      Open;
    end;
    GrpListForm.QuickRep.Preview;
  finally
    GrpListForm.Free;
  end;
end;
```

we restrict rows returned from the database to those where Company begins with the value of the *CustName* parameter.

In the final report (*RptId* = *ManyGrpList*), we've moved the Query component, *RepQuery*, from the report form to *fmDriver* by using cut and paste. We also need to set the various *TQRDBText DataSet* properties appropriately. In resolving the parameter, we need to do some tinkering with the parameter value to convert it to the correct data type:

```
if RptId = 'ManyGrpList' then begin
  ManyGrpForm := TManyGrpForm.Create(Application);
  try
    with fmDriver.RepQuery do begin
      Close;
      ParamByName('pCustNo').AsInteger :=
        StrToInt(ParmsList.Values['CustNo']);
      Open;
    end;
    ManyGrpForm.QuickRep.Preview;
  finally
    ManyGrpForm.Free;
  end;
end;
```

We have covered all the ground needed to develop and test a new report. In summary, the steps are:

1) Start Delphi and open the RptTestDev project.
2) Add a new form.
3) Place a QuickRep component on the form; add Query and/or Table and appropriate QuickReport components to develop the report in the normal manner.

4) Edit the *ExecReport* procedure by adding code to identify the report, resolve parameters, and run the report.
5) Add the report unit name to the **uses** statement in the **implementation** part of the Driver unit (**File | Use Unit**).
6) Run RptTestDev. If you have not tested the report before, you will need to enter values into the report and parameter fields. Use the field information in Figure 1 to determine what values to place in each field.

In most cases, you will need to repeat steps 4, 5, and 6 to refine the report.

The Reports DLL

In the process of developing the reports in RptTestDev, we've done most of the work needed for the report DLL. The Driver unit contains the *ExecReport* procedure, and each report unit contains the code appropriate to the report. The code for the Reports DLL (for the demonstration reports) is simply:

```
library Reports;
uses
  Driver in 'Driver.pas' { fmDriver },
  List in 'List.pas' { ListForm },
  Grplist in 'Grplist.pas' { GrpListForm },
  Manygrp in 'Manygrp.pas' { ManyGrpForm };
{$R *.RES}

exports ExecReport name 'ExecReport';

begin
end.
```

The steps for developing and distributing new or changed reports are:
1) Develop and test the report using RptTestDev.
2) Open the Reports project.
3) Add the new report unit (**Project | Add to Project**).
4) Compile the project (**Project | Compile**).
5) Distribute the Reports.dll file created by the compile step.

Conclusion

Using a DLL for reporting code makes the job of distributing new or changed reports simpler. As described in this article, you can control parameter values and report access. Also, you can provide the user with parameter hints and sample reports. It's a simple technique that should reduce the size of the file needed to distribute new or changed reports. ∆

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\98\OCT\DI9810NK.

Neville Kelly works at Charles Sturt University in Australia as a Programmer/Analyst using an Oracle database and tools. To convince himself that working life does not have to be that hard, he also develops commercial applications from home using Delphi. He may be reached by e-mail at nkelly@csu.edu.au.

By Rod Stephens

Topological Sorting

Ensuring Things Occur in an Orderly Fashion

Many complex projects are made up of a group of interrelated tasks. Some of the tasks are prerequisites for others, but many are completely unrelated. An obvious example is a college course schedule. Math 100 and Math 110 might be prerequisites for Math 120, but you might be able to take Math 100 and Math 110 in either order, or even concurrently.

Another example is building construction. The framing must be finished before the main electrical, plumbing, and roofing work can start, but those three tasks can be performed in any order.

Software development can also involve large sets of tasks that are related in complex ways. To properly build one routine, you may need the output from another. To perform subsystem tests, you may need to feed outputs from one procedure into another, and then feed the results into a third to see that the results make sense. In this case, the first and third procedures must be written before you can properly test the second.

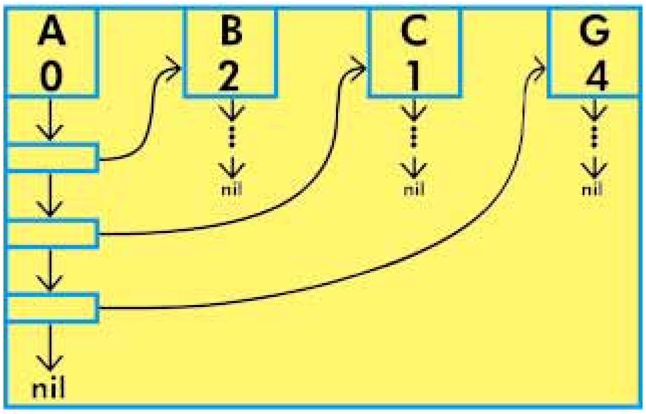


Figure 1: This data structure represents the tasks for which A is a prerequisite.

Orderings like these that specify some, but not all, of the relationships among a group of objects are called *partial orderings*. Extending a partial ordering to produce a full ordering is called *topological sorting*.

Let A < B mean task A must be performed before task B. Then, topological sorting is possible if the relationships among the tasks meet three criteria:

1) Transitivity: For all A, B, and C, if A < B and B < C, then A < C.
2) Asymmetry: For all A and B, if A < B, then B < A is False.
3) Irreflexivity: For all A, A < A is False.

Most real-world examples, like the course prerequisites and construction examples described earlier, obey these rules. However, an example can cause trouble when the relationships are over-specified. For example, suppose Math 100 is required for Math 120; Math 120 is required for Math 10; and Math 10 is required for Math 100. Then, using transitivity, Math 100 < Math 120 and Math 120 < Math 10 implies that Math 100 < Math 10. Combining that result with the fact that Math 10 < Math 100 gives Math 100 < Math 100, which violates irreflexivity. In this example, it's fairly obvious that the courses can't be ordered properly. In a more complicated problem, such as testing a large software project, it may not be as obvious whether there are mutually dependent tasks.

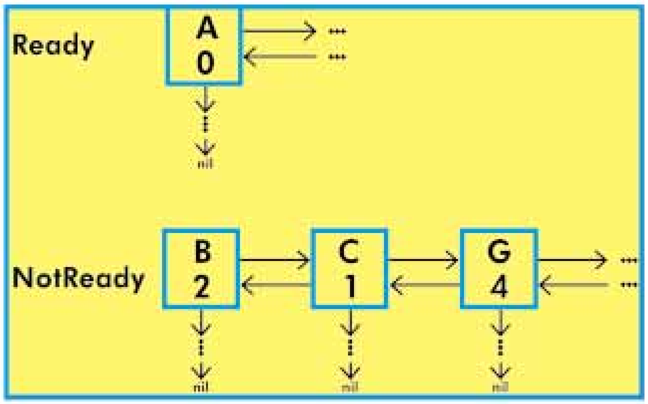


Figure 2: The Ready list contains tasks with no prerequisites. NotReady contains all other tasks.

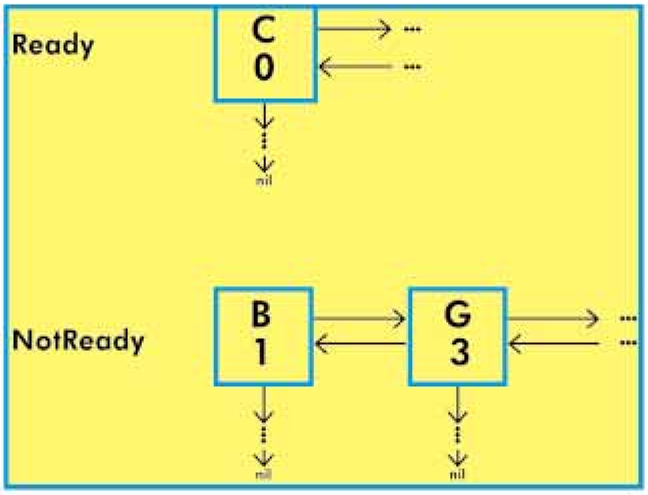


Figure 3: After removing task A, the lists shown in Figure 2 are ready to output the next task.

A Simple Algorithm

There is a very simple method for extending a partial ordering to a complete ordering:

1) Look through the list of dependencies to find a task that has no prerequisites. If there is no such task, the tasks are mutually dependent and can't be ordered.
2) Output that task.
3) Look through the list of dependencies and remove any that have that task on the left. For example, if you just output task A, remove A < B for all B.
4) Repeat step 1 until the dependency list is empty.

For small problems, this algorithm is fine. For large problems, however, it's unnecessarily slow. Suppose there are T tasks and D dependencies. This method repeats steps 1 through 4 a total of T times — once for each task. Steps 1 and 3 take on the order of D steps each. The total time required by this method is on the order of T * D. A more elaborate data structure allows a program to perform topological sorting in the order of T + D time.

A Faster Algorithm

The problem with the simple algorithm is that it examines every dependency in steps 1 and 3, even if a dependency can't possibly be useful. For example, suppose in step 2 that the algorithm outputs task A. In step 3, it must examine all the dependencies, including those like B < C and F < Q, even though task A is not involved in them. The new data structure allows the program to examine only the dependencies that need to be changed because task A was output in step 2.

For each task, create a record that includes the task's name, the number of tasks it has as prerequisites, and a linked list of pointers to the tasks for which it's a prerequisite. For example, suppose task A has no prerequisites. Suppose also that task A is a prerequisite for three other tasks: A < B, A < C, and A < G. Then, the data structure representing task A would look like the diagram shown in Figure 1.

After you have built data structures to represent the tasks, place them in two doubly-linked lists. The first list, Ready, contains tasks that have no prerequisites. These are tasks that can be output immediately. The second list, NotReady, contains all the other tasks. These lists are shown in Figure 2.

The new algorithm follows these steps:

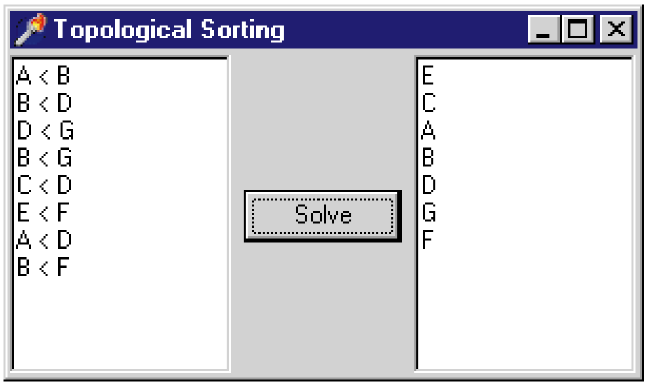
1) Remove the first task from the Ready list.
2) Output that task.
3) Look through this task's linked list of pointers to dependent tasks. For each of those tasks:
   a) Reduce the dependent's prerequisite count.
   b) If the prerequisite count is zero, remove the task from the NotReady list and add it to the Ready list.
4) Repeat from step 1 until the Ready list is empty.

For example, suppose that the program reaches the point shown in Figures 1 and 2. In step 1, it removes task A from the top of the Ready list and outputs it.
The program then searches task A's list of dependents. Figure 1 shows that these include tasks B, C, and G. The program decrements the prerequisite count for each of these tasks, and they become 1, 0, and 3, respectively. At this point, the count for task C becomes 0, so the program removes that task from the NotReady list and adds it to the Ready list. Removing the task is easy because the NotReady list is a doubly-linked list, and it's always easy to remove items from a doubly-linked list. (For more information on linked lists, see Rod Stephens' article "Linked Lists" in the May 1998 *Delphi Informant*.)

Figure 3 shows the Ready and NotReady lists at this point. The program is again ready to output the next item. If there are still tasks in the NotReady list when the program empties the Ready list, those tasks are mutually dependent and cannot be properly ordered.

Delphi Code

The Delphi source code that performs topological sorting is shown in Listing One, beginning on page 21. The *LoadData* procedure reads the dependencies from the Memo control,



```
A < B
B < D
D < G
B < G
C < D
E < F
A < D
B < F
```

```
E
C
A
B
D
G
F
```

Figure 4: The example program, TopoSort, extending a partial ordering.

memInputs. The procedure *OrderData* uses the data structure created by *LoadData* to extend the ordering and display the results in the Memo control, *memOutputs*. Both of these procedures use the procedure *MoveToReady* to move task records from the Ready list to the NotReady list.

The example program, TopoSort, uses these routines to perform topological sorting (see Figure 4). If the tasks can't be ordered, the program lists the tasks that are mutually dependent.

Conclusion

Using the example program TopoSort, you can arrange your course schedule, plan your home construction, and schedule module testing for your biggest software projects. Perhaps equally important, you can determine when a set of tasks is mutually dependent. Then, if someone tells you to perform a group of tasks that can't be properly ordered, you can say with confidence that what they ask is impossible. Δ

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\98\OCT\DI9810RS.

Rod Stephens is the author of several books, including *Custom Controls Library* [1998], *Visual Basic Algorithms* [1998], and *Visual Basic Graphics Programming* [1997], all from John Wiley & Sons. He also writes algorithm columns in *Visual Basic Developer* and *Microsoft Office & Visual Basic for Applications Developer*. He can be reached at RodStephens@vb-helper.com, or see what else he's up to at http://www.vb-helper.com.

Begin Listing One — Topological Sorting

```
type
  DependentPtr = ^Dependent;
  ItemPtr = ^Item;

  // A record to hold a pointer to a dependent item.
  Dependent = record
    DependentItem : ItemPtr;        // The dependent item.
    NextDependent : DependentPtr;  // Next dependent item.
  end;

  // A record to hold an item's dependencies.
  Item = record
    ItemValue       : string[10];    // The item's name.
    NumBefore       : Integer;       // # items before.
    NextItem        : ItemPtr;       // Next item in list.
    PrevItem        : ItemPtr;       // Previous item.
    FirstDependent  : DependentPtr;  // First item after.
  end;

var
  // Sentinels for Ready and NotReady lists.
  NotReadyTop, NotReadyBottom : Item;

// Load the input data from the memInputs Memo control.
procedure TTopoSortForm.LoadData;
var
  i, p : Integer;
  the_line, dep_name, pre_name : string;
  itm, dep_item, pre_item : ItemPtr;
  new_dep : DependentPtr;
begin
  // Initialize the doubly-linked lists.
  NotReadyTop.NextItem := @NotReadyBottom;
  NotReadyTop.PrevItem := nil;
  NotReadyBottom.PrevItem := @NotReadyTop;
  NotReadyBottom.NextItem := nil;
  NotReadyBottom.ItemValue := #255;  // A big value.

  ReadyTop.NextItem := @ReadyBottom;
  ReadyTop.PrevItem := nil;
  ReadyBottom.PrevItem := @ReadyTop;
  ReadyBottom.NextItem := nil;
  ReadyBottom.ItemValue := #255;

  // Get the input data. Load all the items into
  // the NotReady list.
  for i := 0 to memInputs.Lines.Count - 1 do begin
    // Parse this line.
    the_line := Trim(memInputs.Lines[i]);
    if (the_line = '') then
      Continue;

    p := Pos(' ', the_line);
    pre_name := Trim(Copy(the_line, 1, p - 1));
    dep_name := Trim(Copy(the_line, p + 2,
                     Length(the_line)));

    // Find the items.
    pre_item := FindItem(pre_name);
    dep_item := FindItem(dep_name);

    // Add dep_item to pre_item's dependent list.
    // pre_item < dep_item.
    GetMem(new_dep, SizeOf(Dependent));
    new_dep^.DependentItem := dep_item;
    new_dep^.NextDependent := pre_item^.FirstDependent;
    pre_item^.FirstDependent := new_dep;
    dep_item.NumBefore := dep_item.NumBefore + 1;
  end;

  // Move items with no dependencies into the Ready list.
  itm := NotReadyTop.NextItem;
  while (itm <> @NotReadyBottom) do begin
    if (itm.NumBefore > 0) then begin
      // Leave this item in the NotReady list.
      // Prepare to examine the next item.
      itm := itm^.NextItem;
    end
    else begin
      // This item is ready for output.
      // Move it to the Ready list.
      itm := MoveToReady(itm);
    end;
  end; // End while (itm <> @NotReadyBottom) loop.
end;

// Find a complete ordering and display it in memOutputs.
procedure TTopoSortForm.OrderData;
var
```

```
  itm, dep_item : ItemPtr;
  dep           : DependentPtr;
begin
  // While there are items in the Ready list, output one.
  while (ReadyTop.NextItem <> @ReadyBottom) do begin
    // Remove the first item from the Ready list.
    itm := ReadyTop.NextItem;
    ReadyTop.NextItem := itm^.NextItem;
    ReadyTop.NextItem^.PrevItem := @ReadyTop;

    // Add the item to the output.
    memOutputs.Lines.Add(itm^.ItemValue);

    // Decrement NumBefore for all items that are
    // dependent on this one.
    dep := itm^.FirstDependent;
    while (dep <> nil) do begin
      dep_item := dep^.DependentItem;
      dep_item^.NumBefore := dep_item^.NumBefore - 1;
      if (dep_item^.NumBefore < 1) then begin
        // This item has no more dependents.
        // Move it to the Ready list.
        MoveToReady(dep_item);
      end;

      // Free this Dependent record.
      itm^.FirstDependent := dep^.NextDependent;
      FreeMem(dep);

      // Go to itm's next Dependent record.
      dep := itm^.FirstDependent;
    end; // End while (dep <> nil) loop.
  end;  // End while there are items in the Ready list.

  // If there are still items in the NotReady list, they
  // cannot be ordered.
  if (NotReadyTop.NextItem <> @NotReadyBottom) then begin
    memOutputs.Lines.Add('');
    memOutputs.Lines.Add('Mutually dependent:');

    while (NotReadyTop.NextItem <> @NotReadyBottom) do begin
      // Remove the first item from the NotReady list.
      itm := NotReadyTop.NextItem;
      NotReadyTop.NextItem := itm^.NextItem;

      // Add the item to the output.
      memOutputs.Lines.Add(itm^.ItemValue);

      // Remove the dependents for itm.
      dep := itm^.FirstDependent;
      while (dep <> nil) do begin
        // Free this Dependent record.
        itm^.FirstDependent := dep^.NextDependent;
        FreeMem(dep);

        // Go to itm's next Dependent record.
        dep := itm^.FirstDependent;
      end;
    end;  // End while NotReady list is not empty.
  end;  // End if there are items in the NotReady list.
end;

// Move the indicated item to the Ready list. Leave itm
// pointing to the next item in the original list.
function TTopoSortForm.MoveToReady(itm: ItemPtr) : ItemPtr;
var
  after_me, before_me : ItemPtr;
begin
  // This item is ready for output.
  after_me  := itm^.PrevItem;
  before_me := itm^.NextItem;

  // Remove itm from the NotReady list.
  after_me^.NextItem := before_me;
  before_me^.PrevItem := after_me;

  // Add itm to the Ready list.
  itm^.PrevItem := @ReadyTop;
  itm^.NextItem := ReadyTop.NextItem;
  itm^.NextItem^.PrevItem := itm;
  ReadyTop.NextItem := itm;

  // Return the item after itm in its original list.
  Result := before_me;
end;
```

End Listing One

Delphi 2, 3, 4 / Windows NT

By Ted Houts

Writing to the NT Event Log

NT Event Management Made Easy

Often in a Windows NT server environment, system administrators are baffled by proprietary, in-house application initialization files and error logs. By now, most programmers should be using the well-published and documented registry API to localize initialization parameters. However, while a global source for NT error logging exists, it's often left unused. Many applications will log messages to a text file, and others may log to a database table, so there is usually no central and consistent method to view and maintain *all* in-house and third-party system application problems. A typical programmer's implementation of including a menu item to open the application's local log file will not work when implementing a Windows NT service, as most services have no GUI for the user.

NT has had a built-in solution all along. Use of the NT Event Log can consolidate, filter, and group all error messaging in one location, which will reduce training, documentation, and the troubleshooting time of your server applications. Use of NT error logging adds even more value by timestamping the events as they come in, and the log file data can be remotely accessed over a network.

NT has a set of APIs that allow you to programmatically write, read, clear, backup, and respond to changes in the NT Event Log. Unfortunately, you won't find a lot of technical information available to implement NT event logging, especially via Delphi. In fact, Delphi 3 is missing some pieces to do it entirely. Nonetheless, this article will describe the methods to write professional style events to the NT Event Log. (Note: Event logging is an NT feature and is not available in Windows 95.)

Delphi Deficiency

As we all know, the NT API is completely open in Delphi, but the Delphi environment doesn't contain everything required for NT event logging. Along with some missing API constant declarations, there is an add-on executable in the Borland C++ Development Suite that, though not technically required, is a critical piece to creating professional NT Event Log messages. This tool is the Message Compiler, or mc.exe. This console application comes with the Borland C++ Development Suite, and is not part of Delphi 3. This is unfortunate, as Delphi programmers will need this functionality, especially with the continuing popularity of NT.

Terminology

While tackling this topic and reviewing the disparate available documentation, the terminology can be confusing and conflicting. Some of the descriptions may not make sense until you have finished reading this article, but you may find the glossary in Figure 1 useful.

A First Write to the Event Log

Setting aside some of the complexities of the message file resources and registry keys, the listing shown in Figure 2 will write a minimal event entry to the NT Event Log.

Now, open the Event Viewer (if it's already open, refresh it). The Event Viewer can be found from the **Start** menu under **Programs | Administrative Tools | Event Viewer**. From the

Term	Description
NT Event Log	The "global" log file in which NT stores system, security, and application events. Use of the API will allow you to read, write, and clear the data in this log.
NT Event Viewer	An NT administrative tool to view the NT Event Log. From the **Start** menu, go to **Programs | Administrative Tools | Event Viewer** (path is %SystemRoot%\system32\eventvwr.exe).
Event Source	The registry key that defines the message file information your application is using for writing to the NT Event Log. In the examples in this article, the Event Source is MYAPP_REG.
Event Message File	The file that contains the resource-compiled event messages. It's usually a separate DLL (but can be embedded in the resources of primary application) and resides in the path defined in the registry (the Event Source). The path of this file is the value of the *EventMessageFile* entry in the registry. In the examples in this article, the Message File is MYAPP_MSG.DLL.
Category Message File	The file that contains the resource-compiled category messages. It's usually a separate DLL (but can be embedded in the resources of the primary application) and resides in the path defined in the registry (the Event Source). The path of this file is the value of the *CategoryMessageFile* key in the registry. In the examples in this article, the Category Message File is MYAPP_MSG.DLL (the same file as the Event Message File).
Message Compiler	This is the Microsoft tool, mc.exe, described in the Message Compiler section. It compiles the text-based Message Compiler Source Files into a binary format.
EventID	The unique numerical constant defined in the *EventMessageFile* for the particular event you are logging.
CategoryID	The unique numerical constant defined in the *CategoryMessageFile* for the particular event you are logging. If used, this value must be numbered consecutively starting with the number 1.
Message File Resources	The output of the Message Compiler. The resulting files are used to create the Event Message File (DLL).
Message Strings	The strings in the Message File Resources, which will be merged with your additional message information for viewing in the NT Event Viewer (for example, "Started at %1" in eventmsgs.mc).
Message Compiler Source Files	The text file from which the *EventID*, *CategoryID*, and *Message* strings are defined. This file is compiled by the Message Compiler to create the Message File Resources. In this article, the Message File Resource source is eventmsgs.mc.

Figure 1: Glossary of NT Event Log terminology.

```
procedure TFrm_Main.btnLogMinimalClick(Sender: TObject);
var
  hEventLog : THandle;  // Handle to the event log.
  Msg       : PChar;
begin
  // Write directly to the log file, w/o any registry
  // verification.
  Msg := 'Hello NT Event Log';  // A simple message.
  // Server name where message dll resides.
  // nil indicates local machine.
  // ??? is application registry key.
  hEventLog := RegisterEventSource(nil, '???');

  if (hEventLog <= 0) then begin
    raise Exception.Create(
      'RegisterEventSource call failed!');
    Exit;
  end;

  if not ReportEvent(hEventLog, 0, 0, 0,
                     nil, 1, 0, @Msg, nil) then begin
    DeregisterEventSource(hEventLog);
    raise Exception.Create('Failure in Report Event');
  end;

  // Return event log handle retrieved by
  // RegisterEventSource.
  DeregisterEventSource(hEventLog);
end;
```

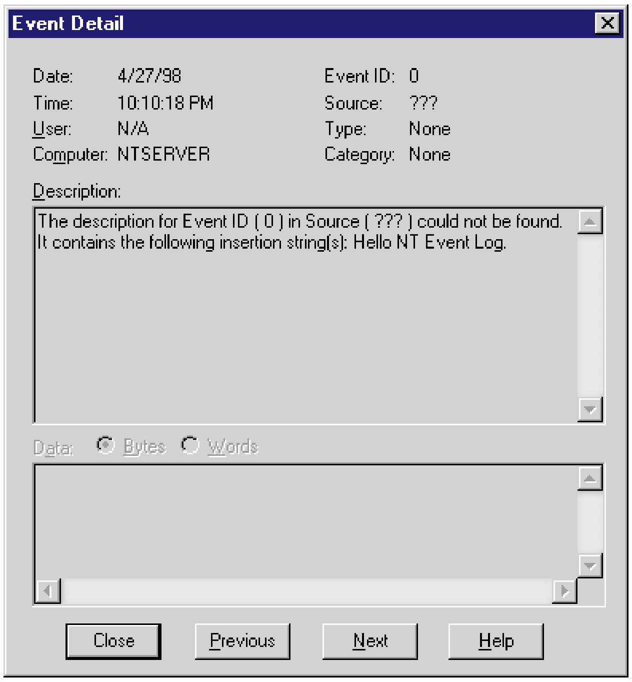
Figure 2: Minimal code necessary to write to the NT Event Log (from frmMain.pas).

main menu of the Event Viewer, choose **Log | Application** as opposed to **System** or **Security** events. Double-click the latest entry to see a dialog box, as shown in Figure 3.

Notice that the code did not include timestamp logic, because NT will automatically save the date and time the message was received. *RegisterEventSource* retrieves a handle to the Event Log, with the first parameter defining the server in which the event source exists (**nil** if local), and the second parameter defining the Event Source for the application. The Event Source is the registry key that, by name, tells the NT Event Viewer where to find the Event Message File to display the details of the message. Because we entered ???, and there is no such key defined, only the raw data we sent to the Event Log is displayed. The NT Event Viewer doesn't know where to look up more information about the event.

The NT Registry, Event Source, and Message File

Typically, messages such as "Couldn't connect to server" or "Invalid packet received," are hard-coded in applications, which are either written to a text log file or displayed in a window. NT has defined a format to keep messages in an external location, which allows you to use them in other applications, reword them to be more informative, or even translate them to a different language. This can be done without rebuilding or redistributing your main application. NT uses a Message File,



```
Date:     4/27/98          Event ID:  0
Time:     10:10:18 PM      Source:    ???
User:     N/A              Type:      None
Computer: NTSERVER         Category:  None
Description:
The description for Event ID ( 0 ) in Source ( ??? ) could not be found.
It contains the following insertion string(s): Hello NT Event Log.
```

Figure 3: The Event Detail dialog box.

```
HKEY_LOCAL_MACHINE
  SYSTEM
    CurrentControlSet
      Services
        EventLog
          Application
            MYAPP_REG
              CategoryCount        0x00000003
              CategoryMessageFile  c:\winnt\system32\myapp_msg.dll
              EventMessageFile     c:\winnt\system32\myapp_msg.dll
              TypesSupported       0x00000007

          Security
          System
```

Figure 4: Registry layout for Event Source definition.

(usually) a DLL that contains a resource with defined event messages to provide these types of features. If these messages are changed and the DLL is copied back in the system, the output of the Event Log is changed. Your original application can be left untouched. NT determines the message files you are using by the definition of the Event Source. Figure 4 shows the registry layout for an Event Source definition.

In this article and the accompanying examples (available for download; see end of article for details), the Event Source is MYAPP_REG, and the Message File is MYAPP_MSG.DLL, which will be deployed to the \System32 directory. The NT Event Viewer reads the keys (Event Sources) under *Application* to fill its filtering and search controls with Event Sources known to the system. Under each of these keys are values that give facts about the particular Event Source. Although it's not covered in this article, a programmer would also need to know these same values to read from the Event Log. The *TypesSupported* value defines the type of messages supported, and 7 is the value to support all the main message types — "Success," "Error," "Information," etc. The *EventMessageFile* and *CategoryMessageFile* values define the path in which the respective file resides on the system. The *CategoryCount* value defines the number of categories defined in the *CategoryMessageFile*, which should be accurate, so that the NT Event Viewer can set up the proper loop-ending value. Also, note that there is a *ParameterMessageFile* value, which is not discussed in this article. Because it's critical that these values are correct, an application writing to the NT Event Log should confirm the configuration. The listing in Figure 5 from the example code assures proper registry configuration. Note that the EVENT_SOURCE, CATEGORY_COUNT, *FMsgDLLPath*, and *FCatDLLPath* are constants or private values of *TEventLog* to assure they are coordinated with

```
procedure TEventLog.CheckRegistry;
var
  Reg: TRegistry;
const
  RegistryMsgRootPath =
 '\SYSTEM\CurrentControlSet\Services\EventLog\Application';
begin
  Reg := TRegistry.Create;
  try
    with Reg do begin
      RootKey := HKEY_LOCAL_MACHINE;
      OpenKey(RegistryMsgRootPath + '\' +
              EVENT_SOURCE, True);
      if ReadString('EventMessageFile') <> FMsgDLLPath then
        WriteString('EventMessageFile', FMsgDLLPath);
      if ReadString(
           'CategoryMessageFile') <> FCatDLLPath then
        WriteString('CategoryMessageFile', FCatDLLPath);

      try
        // Exception on ReadInteger if value is not
        // there. 7 indicates support of all event types.
        if ReadInteger('TypesSupported') <> 7 then
          WriteInteger('TypesSupported',7);
      except on E:Exception do
        WriteInteger('TypesSupported',7);
      end;

      try  // Exception on ReadInteger if no value.
        if ReadInteger(
             'CategoryCount') <> CATEGORY_COUNT then
          WriteInteger('CategoryCount',CATEGORY_COUNT);
        except on E:Exception do
          WriteInteger('CategoryCount',CATEGORY_COUNT);
      end;

      CloseKey;
    end;
  finally
    Reg.Free;
  end;
  // Make sure the files exist on the system, assuming
  // local server. The messages will still log, but the
  // output will be incorrect as viewed through the NT
  // Event Viewer.
  if not FileExists(FMsgDLLPath) then
    raise Exception.Create(
      'Event Message File does not exist!' + Chr(13) +
      Chr(10) + FMsgDLLPath);
  if not FileExists(FCatDLLPath) then
    raise Exception.Create(
      'Category Message File does not exist!' + Chr(13) +
       Chr(10) + FCatDLLPath);
end;
```

Figure 5: Listing of registry configuration confirmation.

```
;//
;// Category Strings - Must start with the number 1.
;//
MessageId=1
Severity=Success
Facility=System
SymbolicName=CATEGORY_ONE_STRING
Language=English
App Start
.

;//
;// Used by TevtAppStart.
;//
MessageId=33
Severity=Success
Facility=System
SymbolicName=MSG_START_TIME
Language=English
Started at %1
.
;// Used by TevtDiskInfo.
;//
MessageId=40
Severity=Success
Facility=System
SymbolicName=MSG_DISK_SPACE
Language=English
Drive Tested: %1
File System: %2
Disk Size: %3
Free Space: %4
Used Space: %5
.
```

Figure 6: Message File Resource from eventmsgs.mc.

actual counts and the location of the external Event Message DLL.

Building an Event Message File

Note that in this article, both the Event Message File and Category Message File will be created as a single DLL, as opposed to binding into the application's executable. Binding the .RES directly in the executable would work (make sure the applicable registry entries point to the path of the application), but would remove the independence features the DLL gives. If another language was desired, or more description messages were requested, a recompile (and of course re-testing) of the application would be required. Depending on the coordination abilities of the local development environment, it may be desirable to use the same Event Message DLL for multiple applications, as many error messages, such as "Application shutdown" and "Unable to connect to database," are common. Building the Events and Categories in a single DLL was done for simplicity, but is not required.

There are three steps to creating the Event Message File:

1) Create the Message Compiler source file.
2) Compile the Message Compiler source file.
3) Create the simple "resource only" DLL by compiling with the Message File Resource.

The source code includes a batch file, make_msg.cmd, which calls the proper executables for the compiling stages.

Create the Message Compiler Source File

The Message Compiler Source file is text-based, conventionally with a file extension of .MC. Besides some header information, the message source file contains blocks of message definition sections containing Name=Value style event message information, and the verbose message string, as shown in Figure 6. Message blocks are terminated by a single period, followed by a new line. No spaces are allowed around the period.

To indicate non-compiled comments, use a semi-colon (;) followed by two forward slashes (//). The value you give to MessageId has some rules. It must be a value from 0 to 65535, and, if defining categories, must be numbered consecutively starting with the number 1. If MessageId is left blank, i.e. MessageId=, the message compiler will automatically set it to the previous value, + 1. The unique string value given to `SymbolicName` will appear in the output C-style header file of the message compilation, and will show the actual value set for the *EventID* or the *CategoryID* to be used in *ReportEvent*.

Set `Severity` to `Success` and `Facility` to `System`, even if you are defining an error message string. The *EventType* value in the *ReportEvent* function — not the *Severity* value set in this file — flags the specific message type in the NT Event Log. The `Language` is set to `English` in the examples, but it's important to note that language independence is one of the primary design goals of the NT Event Log methods. Applications that require multi-lingual support may build a single Event Message File, which contains support for each language set. This article doesn't go into the language specifics, but know that there is built-in support if needed.

The final piece of information is the verbose message strings. These strings are "merged" with the application's array of strings you pass in the eighth parameter of the *ReportEvent* call. The place marker for the first string array element is `%1`, `%2` for the second, etc.

Compile the Message Compiler Source File

The Message Compiler Source File is compiled by running the Message Compiler with the file name as the first parameter, e.g. `mc.exe eventmsgs.mc`. This compilation will generate three new files; these files are used to create the Message File DLL (see Figure 7). One of these files is a C-style header file, which contains the constant definitions for event mes-

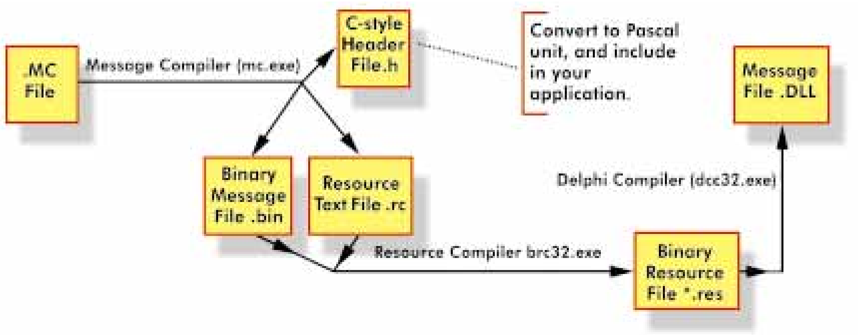

.MC File
Message Compiler (mc.exe)
C-style Header File.h
Convert to Pascal unit, and include in your application.
Message File .DLL
Binary Message File .bin
Resource Text File .rc
Delphi Compiler (dcc32.exe)
Resource Compiler brc32.exe
Binary Resource File *.res

Figure 7: Message files creation flow.

sages. This can easily be converted to a Pascal unit, and included in your project. The other two files are the binary message table and a simple resource file.

Create a Resource-only DLL

Using the Borland Resource Compiler on the .RC file (created by the Message Compiler) creates the .RES to be included in your Event Message DLL project (`brc32.exe -r -foeventmsgs.res -v -32 eventmsgs.rc`). Create a very simple Delphi DLL project, as shown in Figure 8, and run the Delphi Compiler from the command line to create the DLL with the new message resource, `dcc32.exe myapp_msg.dpr`.

At this point, an official Event Message (and Category Message) File is created and ready to be deployed and used by an application. Technically, while much of the difficult work is already done at this point, additional integration issues need to be addressed.

Integration of Event Message File with Applications and Deployment

The relationship of an application logging to the NT Event Log, using a corresponding Event Message File, is very similar to an application utilizing the Windows Help system with an external Help file. When referencing a Help file, the programmer must be provided with the Help topic integer values, which means there must be a close synchronization between the application and Help-file programmers.

The same is true for NT Event Log development; the programmer must know the event messages and defined values. An attempt to create a single generic Delphi component to encapsulate the NT Event Logging would be very difficult, as each message will require different input than others. However, some sort of component-based strategy allows easy sharing of the Event Message DLL and API throughout multiple applications. A logical development assignment would be that one developer would create the Event Message File, and build "wrapper" components for the events, and the other developer would use the components in an application. The implementation used in the example source is to create a component, *TEventLog*, that primarily simplifies the NT Event Log API and registry specifics, and then create a set of descendant components, one for each specific event message. *TEventLog* has a private method, *CheckRegistry*, which is called before writing to the Event Log to assure the registry is properly configured (as described earlier in this article). Its other primary task is to simplify the *ReportEvent* API call. *ReportEvent* returns True if the entry was written to the log, and is declared in Windows.pas as follows:

```
function ReportEvent(hEventLog: THandle;
  wType, wCategory: Word; dwEventID: DWORD;
  lpUserSid: Pointer; wNumStrings: Word; dwDataSize: DWORD;
  lpStrings, lpRawData: Pointer): BOOL; stdcall;
```

where:

- *hEventLog* is the handle returned by *RegisterEventSource.*
- *wType* is the type of event to be logged.

```
library myapp_msg;

{$R eventmsgs.RES}  // Resource with event message info.
begin
end.
```

Figure 8: Simple DLL, from myapp_msg.dpr.

```
Const Value
(found in winnt.h)
Meaning
NT Event Viewer String
NT Event Viewer Icon Displayed
EVENTLOG_SUCCESS = $0000
Success
Success
Blue Circle with white "I" character
EVENTLOG_INFORMATION_TYPE = $0004
Information
Information
Blue Circle with white "I" character
EVENTLOG_WARNING_TYPE = $0002
Warning
Warning
Yellow Circle with black "!" character
EVENTLOG_ERROR_TYPE = $0001
Error
Error
Red Stop Sign

EVENTLOG_AUDIT_SUCCESS = $0008
Success Audit
Success Audit
Yellow key icon

EVENTLOG_AUDIT_FAILURE = $0010
Failure Audit
Failure Audit
Padlock icon
```

Figure 9: NT Event Log type values.

- *wCategory* is the category identifier defined in the Category Message File. Set to 0 for a default category description of NONE.
- *dwEventID* is the event identifier defined in the Event Message File.
- *lpUserSid* is the user's security identifier (not discussed in this article). Set to **nil** if not required.
- *wNumStrings* is the number of null-terminated strings in the *lpStrings* parameter array.
- *dwDataSize* is the number of bytes of raw data to write to the log, as pointed to by *lpRawData*. This value is shown in the detail view of the NT Event Log viewer. Set to 0 if not sending raw data.
- *lpStrings* is an array of null-terminated strings, which are application set values.
- *lpRawData* is the raw data buffer. Set to **nil** if not sending raw data.

Figure 9 shows the constant value declared in winnt.h. Also shown is what appears in the NT Event Viewer for the specific value.

The trickiest part of *TEventLog* is encapsulating the *lpStrings* parameter. *ReportEvent* expects this to be a pointer to an array of null-terminated strings. The first string will be inserted in the %1 marker, as defined in the Message Compiler Source File, the second in %2, etc. Because working with *TStringList* is so familiar, the descendant classes of *TEventLog* can use the *Msg* property to

```
constructor TEvtAppStart.Create(AOwner: TComponent);
begin
  inherited Create(AOwner);
  // Code these protected values to assure they are
  // "in sync" with Message File DLL.
  EventType  := etInformation;
  EventID    := MSG_START_TIME;
  CategoryID := CATEGORY_APP_START;
end;
```

Figure 10: Setting up values for a specific message.

```
procedure TFrm_Main.btnLogDiskInfoClick(Sender: TObject);
begin
  EvtDriveInfo1.Msg.Clear;
  with GetDiskSpaceInfo(cbDrives.Text[1]) do begin
    // Fill msg with drive info.
    EvtDriveInfo1.Msg.Add(Drive);
    EvtDriveInfo1.Msg.Add(FileSystem);
    EvtDriveInfo1.Msg.Add(DiskSize);
    EvtDriveInfo1.Msg.Add(FreeSpace);
    EvtDriveInfo1.Msg.Add(UsedSpace);
    EvtDriveInfo1.Msg.Add(PercentFull);
  end;
  EvtDriveInfo1.Log;
  MessageDlg('Disk info sent to NT Event Log!',
             mtInformation, [mbOK], 0);
end;
```

Figure 11: Using *TevtDiskInfo*.

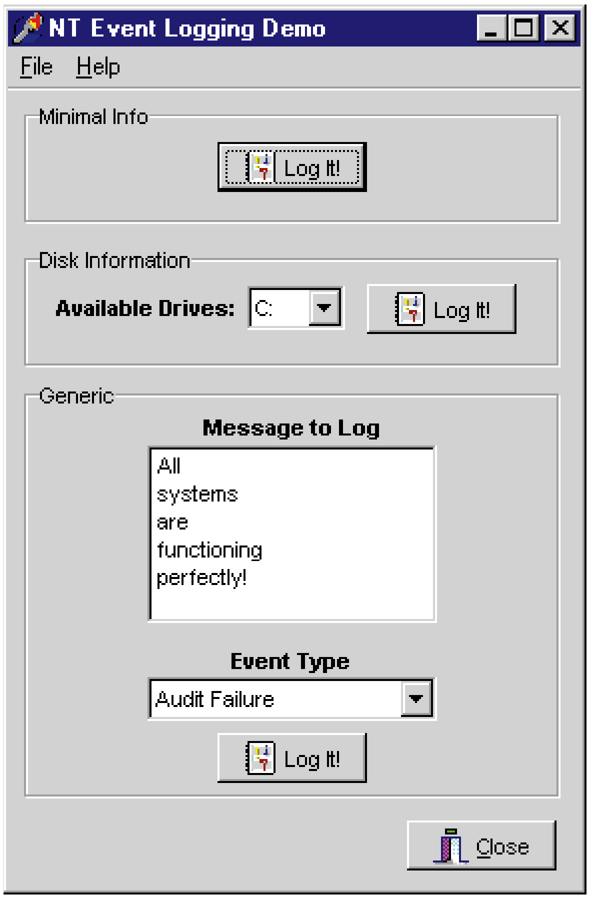


Figure 12: NT Event Log demonstration application.

fill these strings with the well-known methods *Add*, *Insert*, etc. *SetMsg* is called before calling *ReportEvent*, which will format the *TStringList* to an array of PChar, and set the *wNumStrings* parameter, which is simply the *TStringList.Count* value.

For the specific message indicating the start time of the application, *TEvtAppStart* is created, which descends from *TEventLog*. The *TEvtAppStart* component hard-codes the *EventID* and *CategoryID* for the message, wrapping it up as shown in Figure 10.

Note that the constants *EventID* and *CategoryID* are assigned to a constants-defined header file created by the Message Compiler. An application can drop this component on the main form, and in the *FormCreate* procedure, call the *Log* method, with no additional work. Suppose a requirement is to log disk-drive information when an application starts. A disk information message is created in Event Message File, which expects five data values (again, see Figure 6).

A component, *TevtDiskInfo*, is created — again descending from *TEventLog*, which, in its *Log* event, will format the values and write to the Event Log. In this case, the application programmer will need to know the order of the strings, but does not need to know the event message constants wrapped in the component.

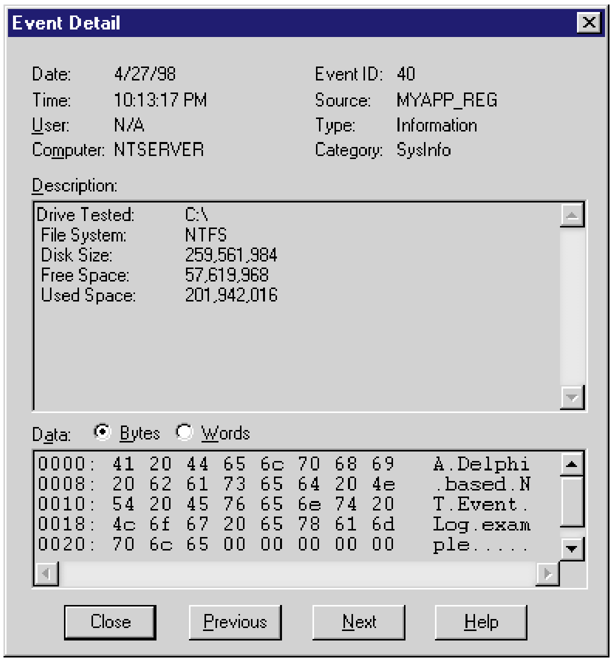
The listing in Figure 11 calls a utility function, *GetDiskSpaceInfo*, which, when passed a drive letter, returns a record of disk information. The code adds the values to *Msg* and calls *Log*. As long as the Event Message File is deployed on the target machine, these components can be used by many different applications to write to the NT Event Log. Make sure that whatever the deployment strategy, the Event Message File is copied to the proper directory on the target machine. Most applications put the file in the \System32 directory, but this is not required; it doesn't matter, as long as the directory path is the same as expected by the application.

Demonstration Application

The demonstration application shows the use of these custom components, and can demonstrate the effect of the *EventType* values on the NT Event Viewer (see Figure 12).

Entries are made to the NT Event Log when the application is created and destroyed. A combo box is filled with available drives, and with the click of a button, will write the information to the NT Event Log using the *TEvtDiskInfo* component. Figure 13 shows the event detail from the NT Event Viewer. Note that *TEventLog* put in dummy data "A Delphi based NT Event Log example," but this data could be used to store internal codes or OS error return codes.

A generic message event was created for free-form messages to demonstrate various error types. A combo box contains various error types, and a memo to enter a message. Figure 14 shows the NT Event Viewer GUI with the demonstration messages with different error types. Generic messages are not recommended for real applications, as the Event Message File



Event Detail

Date: 4/27/98 Event ID: 40
Time: 10:13:17 PM Source: MYAPP_REG
User: N/A Type: Information
Computer: NTSERVER Category: SysInfo

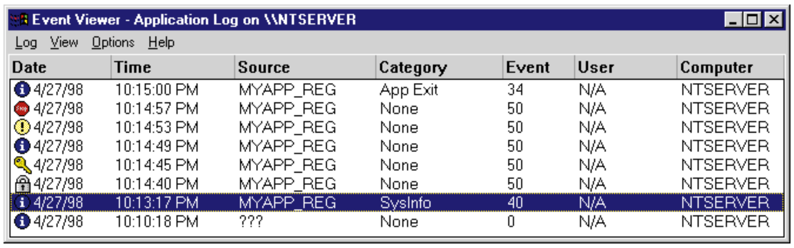
Description:

```
Drive Tested:     C:\
 File System:     NTFS
 Disk Size:       259,561,984
 Free Space:      57,619,968
 Used Space:      201,942,016
```

Data: [✓ Bytes] Words

```
0000: 41 20 44 65 6c 70 68 69   A.Delphi
0008: 20 62 61 73 65 64 20 4e   .based.N
0010: 54 20 45 76 65 6e 74 20   T.Event.
0018: 4c 6f 67 20 65 78 61 6d   Log.exam
0020: 70 6c 65 00 00 00 00 00   ple.....
```

Figure 13: NT Event Viewer detail of disk information.



Event Viewer - Application Log on \\NTSERVER

Date	Time	Source	Category	Event	User	Computer
4/27/98	10:15:00 PM	MYAPP_REG	App Exit	34	N/A	NTSERVER
4/27/98	10:14:57 PM	MYAPP_REG	None	50	N/A	NTSERVER
4/27/98	10:14:53 PM	MYAPP_REG	None	50	N/A	NTSERVER
4/27/98	10:14:49 PM	MYAPP_REG	None	50	N/A	NTSERVER
4/27/98	10:14:45 PM	MYAPP_REG	None	50	N/A	NTSERVER
4/27/98	10:14:40 PM	MYAPP_REG	None	50	N/A	NTSERVER
4/27/98	10:13:17 PM	MYAPP_REG	SysInfo	40	N/A	NTSERVER
4/27/98	10:10:18 PM	???	None	0	N/A	NTSERVER

Figure 14: The NT Event Viewer with application messages.

should contain most of the verbose text, so that it can be translated or rewritten. Note how the NT Event Viewer extracted the category string from the event file for display, as the *ReportEvent* call only takes an integer.

Conclusion

At some point, you'll need to write your application messages to the NT Event Log. This practice should be encouraged for the benefit it brings to the users (e.g. IT personnel), as well as assisting in maintenance and field troubleshooting of your applications. While grasping the methods of the NT Event Log is slightly advanced, it can become simplified by building custom components to hide the unique values and API complexities.

While this article explains the methods of writing to the NT Event Log is, there are additional APIs available to clear, read, backup, and receive notifications when a write to the log has taken place. Using these other features, one could create a custom NT Event Viewer that filters only specific application messages, displays the Event Log entries on a Web page, or sends an e-mail when a particular message is encountered. ∆

The files referenced in this article are available on the Delphi Informant Works CD located in INFORM\98\OCT\DI9810TH.

Ted Houts is a client/server software development contractor currently doing work at a large telecommunications company. He has been developing in Delphi since its initial release. Ted can be reached via e-mail at thouts@rockway.com.

By Cary Jensen, Ph.D.

Delphi Database Development

Part II: Tables, Queries, and Stored Procedures

The previous installment of "DBNavigator" (in the August, 1998 *Delphi Informant*) was the first article in a series intended to re-examine Delphi database development. That column explored the relationship between the various data access components and the Borland Database Engine (BDE). The series continues this month with a look at the basic configuration and use of the *TDBDataSet* classes, including *TTable*, *TQuery*, and *TStoredProc*.

In Delphi 2 and earlier, the classes discussed here were referred to as *TDataSet* classes. In Delphi 3 Client/Server, the *TDataSet* class also includes the *TClientDataSet* class (a detailed discussion of which will be deferred to a future article). The distinction is that the *TDBDataSet* classes make use of the BDE, while the *TClientDataSet* does not (it, instead, relies on the DBClient DLL for data access). While the Table, Query, StoredProc, and ClientDataSet components are all *TDataSet* descendants, only the first three descend from *TDBDataSet*.

As mentioned at the outset, the following discussion focuses on basic configuration and use of these classes. However, before discussing how to access data using *TDBDataSet* classes, it's worth taking a moment to consider the origin of the data that these components access.

Tables, Data, and Files

The data used in the examples that normally appear in this column can be found in Delphi's \Demos\Data subdirectory. These database files were installed when you installed Delphi. While these files are perfect for a demonstration, you'll use other files when creating your own databases. This raises a fundamental question: Where do you get your data files?

There are a variety of answers, and they range from the simple to the complex. In some instances, you are provided with existing files that contain the data that will be employed by the application. This is often the case when you are building an application in Delphi based on an existing application (such as creating a Windows version of an old DOS or mainframe application). This situation is even easier when Delphi directly supports the file type of the tables you are provided, such as when you're creating a Delphi application based on a pre-existing Paradox for Windows application or dBASE application. All but the earliest Paradox and dBASE files can be used in Delphi applications without modification.

If you're provided with files in a format not supported directly by Delphi, you must first convert these files into a supported format. How difficult the conversion will be depends on the original file type. For example, if the data is in a delimited ASCII format, there are a number of tools for converting these files to dBASE, or some other useful file format. In a worst-case scenario, you will need to manually write a data translation program. (This can be done in Delphi using the standard Pascal I/O functions.) Note, however, that unsupported file formats are not common, due to Delphi's support for ODBC (Open Database Connectivity) drivers, which exist for most database file formats.

If data doesn't exist, you will be responsible for creating the tables yourself. If the application specification includes the specifications for the tables, you can use the Database Desktop that ships with Delphi to create the table structures. (A table structure defines the number of fields [columns] in the table, as well as their names and data types.)

If no definition of the tables required by the application exists, you will either need to define these yourself, or enlist an experienced database developer or database analyst to help you do so. Creating a specification involves identifying the data that needs to be stored, and designing table structures that permit it to be stored correctly and efficiently. There are a number of books on the market that describe this process.

Basic *TDBDataSet* Configuration

Regardless of which DBDataSet component you use, you must provide it with a minimum of two pieces of information. The first is the type of driver to use for data access. In the case of local databases, the BDE uses this driver to read and write the data. For remote database servers, the driver defines how the BDE communicates with the database server.

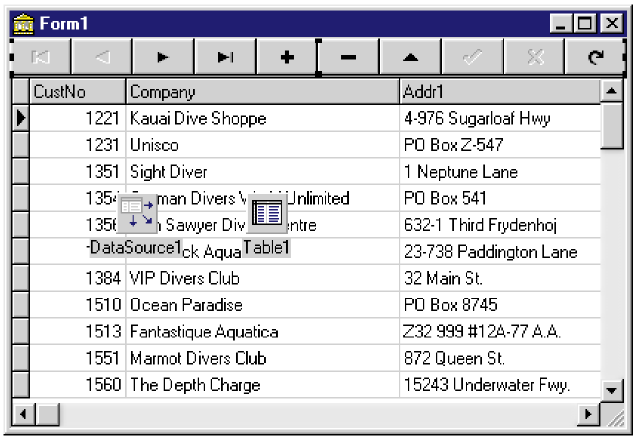
The second piece of information required for data access is a reference to the data itself. For example, when using local Paradox tables, a DBDataSet must know which table to access. Likewise, when using a remote server, it's necessary to indicate which tables, views, or stored procedures you want to use.

Each of the DBDataSet components have different properties for defining these two basic pieces of data. These are described separately in the following sections.

Accessing Data Using Tables

A Table component is the most general — and easiest to use — of the DBDataSet components. It can be used to access data in any BDE-supported format. For example, you can use a Table component to read and write from Paradox and Oracle tables.

The basic access to data with a Table component involves, at a minimum, a single property, but more often two properties. When you are accessing local tables, you merely need to define the *TableName* property. This property can contain either the



CustNo	Company	Addr1
1221	Kauai Dive Shoppe	4-976 Sugarloaf Hwy
1231	Unisco	PO Box Z-547
1351	Sight Diver	1 Neptune Lane
135[illegible]	[illegible]man Divers [illegible] Unlimited	PO Box 541
135[illegible]	[illegible] Sawyer Div[illegible]ntre	632-1 Third Frydenhoj
[illegible]	[illegible]ck Aqua[illegible]	23-738 Paddington Lane
1384	VIP Divers Club	32 Main St.
1510	Ocean Paradise	PO Box 8745
1513	Fantastique Aquatica	Z32 999 #12A-77 A.A.
1551	Marmot Divers Club	872 Queen St.
1560	The Depth Charge	15243 Underwater Fwy.

Figure 1: Accessing data using a Table component.

name of a Paradox or dBASE table, or it can contain the fully-qualified DOS path and table name. Including the DOS path in the *TableName* property is required if you want to activate the table at design time. If you don't need design-time access to data, you can enter only the table name, and the BDE will look for that table in the same directory as your project's EXE.

Normally, the BDE will determine which driver to use to access your data based on the *DatabaseName* property. However, when you do not assign a value to this property using the *TableName* property alone, the BDE will determine which driver to use (specifically, which of several available BDE DLLs to use) based on the file extension of the table name. If the file extension is omitted, it will choose a driver based on the *TableType* property.

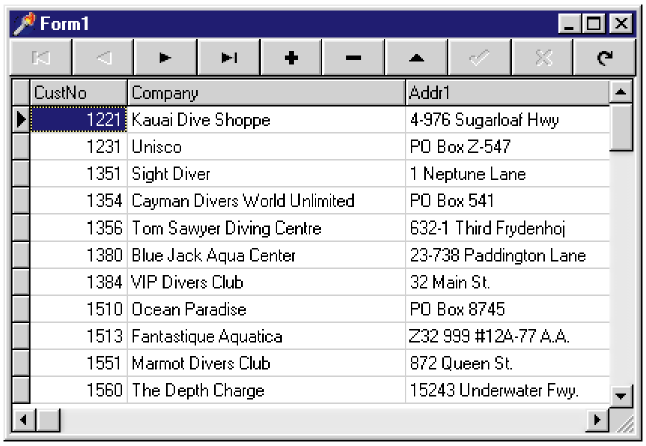
As previously mentioned, the BDE can determine which driver to use, based on the *DatabaseName* property. This is the second of the two properties typically used to configure a Table component (again, the first being *TableName*). Assign one of three possible values to this property. First, you can assign any configured ODBC data source name. A configured ODBC driver includes a data source location. When you do so, the BDE will use the ODBC socket to communicate with the specified driver, converting all instructions to Structured Query Language (SQL) statements.

The second type of value you can assign to the *DatabaseName* property is the name of a database configured using the BDE Administrator. A configured database is often referred to as a global alias because it can be used by any BDE-aware application. Among other things, a configured global alias defines both the driver and data location.

The third type of value you can assign to the *DatabaseName* property is the value of a *DatabaseName* property of a Database component within your project. This type of value is called a local alias because it's available only within the project in which the Database component appears. To use a local alias, the Database component must be configured. This involves defining the driver type and data location. Configuring a Database component will be covered in a future "DBNavigator" article.

These steps demonstrate the configuration of a Table component:

1) Create a new project. Add to it one DBNavigator, one DBGrid, one DataSource, and one Table component.
2) Set the DBNavigator's *Align* property to *alTop*, and the DBGrid's *Align* property to *alClient*. Next, set the *DataSource* property of both the DBNavigator and DBGrid to *DataSource1*.
3) Set the DataSource's *DataSet* property to *Table1*.
4) Now it's time to configure the Table. Begin by setting its *DatabaseName* property to *DBDEMOS*. This is a global alias that was created during Delphi's installation. Next, set the Table's *TableName* property to CUSTOMER.DB. If you now set the Table's *Active* property to True, you'll see the design-time view of your data in the DBGrid, as shown in Figure 1. In addition, it ensures that the Table is opened when the form is created at run time.



CustNo	Company	Addr1
1221	Kauai Dive Shoppe	4-976 Sugarloaf Hwy
1231	Unisco	PO Box Z-547
1351	Sight Diver	1 Neptune Lane
1354	Cayman Divers World Unlimited	PO Box 541
1356	Tom Sawyer Diving Centre	632-1 Third Frydenhoj
1380	Blue Jack Aqua Center	23-738 Paddington Lane
1384	VIP Divers Club	32 Main St.
1510	Ocean Paradise	PO Box 8745
1513	Fantastique Aquatica	Z32 999 #12A-77 A.A.
1551	Marmot Divers Club	872 Queen St.
1560	The Depth Charge	15243 Underwater Fwy.

Figure 2: Data displayed in a DBGrid component on a running Delphi form.

If design-time access to your data isn't necessary, you can leave the Table's *Active* property set to False, and add the following code to the form's *OnCreate* event handler:

```
Table1.Open;
```

Calling the Table's *Open* method has the same effect as setting its *Active* property to True. Press F9 to compile and run the form. Your screen should now look like that in Figure 2.

Basic Query Configuration

Use the Query component to execute SQL statements against one or more databases. These SQL statements can either be Data Definition Language (DDL) statements (e.g. CREATE TABLE and ALTER INDEX), or they can be Data Manipulation Language (DML) statements (e.g. SELECT, UPDATE, and DELETE). The most common statement, however, is the SELECT statement, which produces a view similar to that available using a Table component.

The minimum configuration of a query involves setting its SQL property. For example, the following SQL statement returns all fields and records (columns and rows) from the Customer table in the directory pointed to by the DBDEMOS alias:

```
SELECT * FROM ":DBDEMOS:Customer"
```

Alternatively, if you set the *DatabaseName* property of the Query to *DBDEMOS*, you can omit the alias name and quotation marks from the SQL statement. For example, if you set *DatabaseName* to *DBDEMOS*, the following SQL statement is equivalent to the preceding one:

```
SELECT * FROM Customer
```

After you set the SQL property, and optionally, the *DatabaseName* property, setting the *Active* property to True causes the query to be executed. (A query, such as DELETE, that doesn't return a result set, is executed by calling *ExecSQL*.)

If your query uses a SELECT statement, it's sometimes possible to edit the result set. Two conditions must exist to edit the result set of a query. The first is that the query's *RequestLive* property must be set to True. Second, the result set must contain records from a single table, and there must be a one-to-one correspondence between the records in the result set and the queried table. In other words, if the query contains a join between two tables, or makes use of statements such as DISTINCT or SUM, the query can't be edited (unless the query's *CachedUpdates* property is set to True). Cached updates will be discussed in detail in a future "DBNavigator" article.

You can test whether a query is live (editable) or not by calling its *CanModify* method. If *CanModify* returns True, the BDE will issue a corresponding UPDATE, INSERT, or DELETE query against the table you queried in response to posting changes, and inserting or deleting records, respectively.

Preparing Queries

Before a query can be executed against a database server, it must be *prepared*. The process of preparation involves generating an execution plan on the remote database server. If you open or execute a query without preparing it, the BDE will automatically prepare the query prior to execution, and unprepare it when the query is closed.

Preparing a query is a time-consuming process. If you need a query to execute very quickly when it's opened, you should explicitly prepare the query in advance. One common event used for performing this operation is the *OnCreate* event handler for the form or data module on which the query appears. When you explicitly prepare a query, the BDE doesn't "unprepare" it for you. Instead, you must explicitly call *Unprepare* to release the resources on the database server. This can easily be done from the *OnClose* event handler for the form or data module.

Explicitly calling *Prepare* is particularly important when you are working with parameterized queries. A parameterized query includes one or more parameters. A parameter is a label, similar to a variable, that must be assigned data before the query's execution. A parameterized query doesn't need to be re-prepared each time the parameter is changed. This permits you to execute a parameterized query repeatedly, without the overhead of preparation. If you fail to explicitly call *Prepare* on a parameterized query, however, the BDE will prepare and unprepare the query each time you open and close the query (you must close a query before changing its parameter).

A Query Example

Use the following code to demonstrate the use of a Query component. This example makes use of the Local InterBase Server (LIBS). If you are using Delphi 3, installing LIBS is a separate process from the installation of Delphi. If you currently don't have LIBS installed, you must do so first before continuing with this example. Make sure to restart your computer after installing LIBS, as it's automatically loaded when you boot your system.

If you are using Delphi Standard Edition, you don't have LIBS. Still, you can follow along with this example using a local Paradox table, but the query will be executed by the BDE, rather than a remote database server. To use a Paradox table, set the Query's *DatabaseName* property to *DBDEMOS* in step 4, instead of *IBLOCAL*:

1) Create a new project. Add to it one DBNavigator, one DBGrid, one DataSource, and one Query component.
2) Set the DBNavigator's *Align* property to *alTop*, and the DBGrid's *Align* property to *alClient*. Next, set the *DataSource* property of both the DBNavigator and DBGrid to *DataSource1*.
3) Set the DataSource's *DataSet* property to *Query1*.
4) Now you're ready to configure the Query. Set its *DatabaseName* property to *IBLOCAL*, and its SQL property to `SELECT * FROM Customer`. Next, set the *RequestLive* property to True.
5) You should now add code to control the Query's preparation. Add an *OnCreate* handler to the form and add the following code to it:

```
if Query1.Active then
  Query1.Close;
Query1.Prepare;
Query1.Open;
```

6) Finally, create an *OnClose* event handler for the form and add the following code to it:

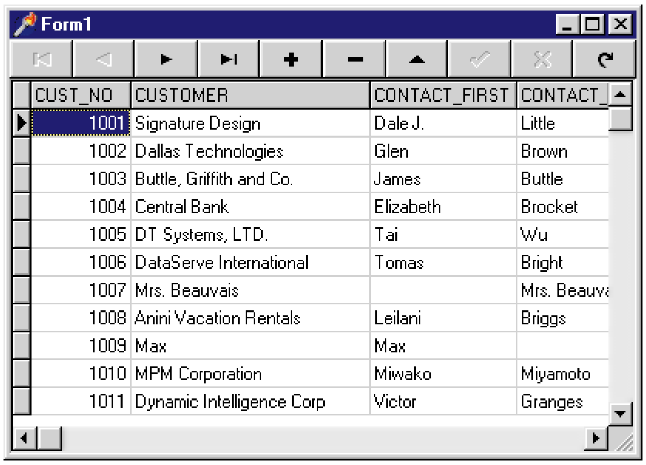
```
if Query1.Active then
  Query1.Close;
Query1.Unprepare;
```

7) Press F9 to run the form. Once the form is created, the query is prepared and executed. Executing the query requires access to the server. Because this access requires a password, a password dialog box is displayed. Enter the password `masterkey` in the **Password** field of the Database Login dialog box (this password is case-sensitive). After you have accepted the password dialog box, your form is displayed, as shown in Figure 3.

Tables vs. Queries

Because Query and Table components can be used interchangeably to edit data, an obvious question is: Which one should you use in your applications? In most cases, the answer to this question is straightforward. When accessing local Paradox and dBASE tables, a Table component generally provides faster access. On the other hand, when you're using data on a remote database server, you'll typically get better performance if you use a Query component.

This is a rule of thumb, however; it may not apply in all situations. If performance is your primary concern, it would be a good idea to test the relative performance for your particular



CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_
1001	Signature Design	Dale J.	Little
1002	Dallas Technologies	Glen	Brown
1003	Buttle, Griffith and Co.	James	Buttle
1004	Central Bank	Elizabeth	Brocket
1005	DT Systems, LTD.	Tai	Wu
1006	DataServe International	Tomas	Bright
1007	Mrs. Beauvais		Mrs. Beauva
1008	Anini Vacation Rentals	Leilani	Briggs
1009	Max	Max	
1010	MPM Corporation	Miwako	Miyamoto
1011	Dynamic Intelligence Corp	Victor	Granges

Figure 3: A live query result set can be edited like a table.

data. On the other hand, you might consider using a Table with a remote database if you're more comfortable having the BDE generate your SQL, rather than writing it manually. In fact, if your database is indexed properly, and you make use of Table methods that leverage indexes, such as *Locate*, *FindKey*, *FindNearest*, and *SetRange*, a Table component's performance against remote database servers can be more than satisfactory.

Using Stored Procedures

Unlike Table and Query components, which can be used with any database (local or remote), StoredProc components can only be used with remote database servers. Stored procedures are pre-compiled subroutines that reside on the server. A stored procedure can consist solely of SQL statements, but can also include additional statements defined by the server's compiler. For example, stored procedures can include control structures to provide program logic on the server.

There are two primary advantages of stored procedures. First, because they reside on the server, they can be used by two or more applications to consistently perform a particular operation. For example, a stored procedure to add a new employee to a database ensures that all applications that use it insert a new employee correctly.

The second advantage — and the principle reason stored procedures are so attractive — is that they provide the most efficient means of data manipulation. Stored procedures can be executed without having to send lengthy SQL statements to the server. In addition, because stored procedures are pre-compiled, they can be executed with less overhead than an equivalent SQL query. Finally, because stored procedures can include statements beyond SQL, they can be used to leverage the strengths of the database server on which they are defined.

Delphi permits you to execute stored procedures, but not to write them. Instead, you use the tools associated with your database server to write and compile your stored procedures. Consequently, it goes without saying that you can create stored procedures only if you have sufficient access rights to the server on which your data resides.

There are two general types of stored procedures: those that return a cursor to a result set (similar to a SELECT query), and those that do not. Stored procedures that don't return a result set may perform some operation on the server, such as emptying a table, or they may return single values by means of output parameters. In fact, both kinds of stored procedures can make use of parameters that permit you to send information to the stored procedure. For example, a stored procedure used to archive old records may be passed a date that it will use to determine which records are old.

Whether a stored procedure returns a dataset may influence which component you use to execute the stored procedure. For example, when executing a stored procedure on an InterBase server, you can use a StoredProc component only if the stored procedure doesn't return a result set. You must use a Query component for those InterBase stored procedures that return a result set. On the other hand, if you're using stored procedures in Microsoft SQL Server database, you can always use the StoredProc component, regardless of whether the routine returns a result set.

Like queries, stored procedures must be prepared prior to their execution. In fact, you can apply the same principles concerning the preparation of queries to stored procedures. Specifically, if you don't explicitly prepare a stored procedure, Delphi will do it for you. Furthermore, stored procedures prepared automatically are also unprepared automatically when they are closed. However, if you explicitly prepare a stored procedure by calling its *Prepare* method, Delphi will not unprepare it, meaning that you'll also be responsible for calling the StoredProc's *Unprepare* method. As a result, it's generally considered a good idea to explicitly prepare a stored procedure that's called repeatedly, explicitly calling its *Unprepare* method when you no longer need it.

Stored Procedure: Example 1

This first example demonstrates how to execute a stored procedure that includes input and output parameters, but doesn't return a result set. For this purpose, we will use the stored procedure named MAIL_LABEL, which is available in the EMPLOYEE.GDB database that's installed with LIBS. Consequently, to follow along with this example, you must have LIBS installed (you cannot execute stored procedures with local tables):

1) Begin by creating a new project. Place two Panel components onto it. Set the *Caption* property of both panels to an empty string. Next, set the *Align* property of one of the panels to *alRight*, and the *Align* property of the other to *alClient*.
2) Place one DBNavigator and one DBGrid into the Panel aligned to the client. Align the DBNavigator to *alTop*, and the DBGrid to *alClient*.
3) Next, place six Label components into the Panel aligned to the right. Adjust their position so they're left-aligned with respect to one another, and are spaced equally. Their positioning should resemble the lines of a correctly-addressed letter. Your form should look something like Figure 4.
4) Now add two DataSource components to the form: one Query and one StoredProc. Set the *DataSet* property of DataSource1 to *Query1*, and the *DataSet* property of *DataSource2* to *StoredProc1*.
5) Select *Query1* and set its *DatabaseName* property to *IBLOCAL*, and its *SQL* property to `SELECT * FROM Customer`.
6) Select *StoredProc1* and set its *DatabaseName* property to *IBLOCAL*, and its *StoredProcName* property to *MAIL_LABEL*.
7) Select both the DBNavigator and the DBGrid, and set their *DataSource* properties to *DataSource1*.
8) It's now time to add code to control the preparation of the stored procedure. (In this case, we won't prepare the query, because it will be executed only once.) Select *Form1* in the Object Inspector, and double-click the space to the right of the *OnCreate* event handler on the Events page. Enter the following code into the method generated by Delphi:

```
StoredProc1.Prepare;
Query1.Open;
```

   This code prepares the stored procedure and then opens (executes) the query.
9) Because we explicitly prepared the stored procedure, we must also unprepare it. With *Form1* still selected, double-click the *OnClose* event property in the Object Inspector, and enter the following code:

```
StoredProc1.Unprepare;
```

10) The execution of the stored procedure will be performed each time the user navigates to a new record in the query. This can be achieved by adding the following code to the *OnDataChange* event handler of DataSource1. Select

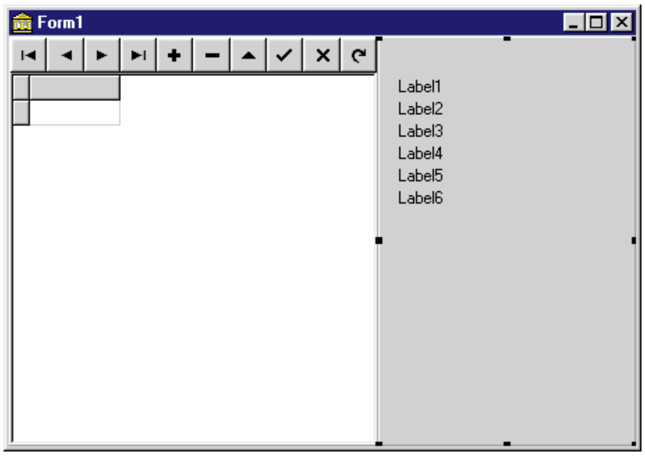
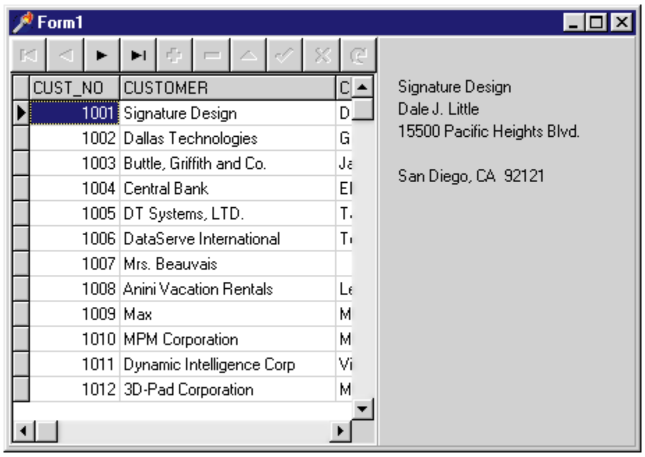


Figure 4: The stored procedure example under construction.



CUST_NO	CUSTOMER	C
1001	Signature Design	D
1002	Dallas Technologies	G
1003	Buttle, Griffith and Co.	Ja
1004	Central Bank	El
1005	DT Systems, LTD.	T.
1006	DataServe International	T
1007	Mrs. Beauvais	
1008	Anini Vacation Rentals	Le
1009	Max	M
1010	MPM Corporation	M
1011	Dynamic Intelligence Corp	Vi
1012	3D-Pad Corporation	M

Signature Design
Dale J. Little
15500 Pacific Heights Blvd.

San Diego, CA 92121

Figure 5: The mailing label is formatted by a stored procedure on the Local InterBase Server.

DataSource1, then double-click the *OnDataChange* event property in the Object Inspector. Add the following code to the generated method:

```
StoredProc1.ParamByName('CUST_NO').AsString :=
  Query1.FieldByName('CUST_NO').AsString;
StoredProc1.ExecProc;
Label1.Caption :=
  StoredProc1.ParamByName('Line1').AsString;
Label2.Caption :=
  StoredProc1.ParamByName('Line2').AsString;
Label3.Caption :=
  StoredProc1.ParamByName('Line3').AsString;
Label4.Caption :=
  StoredProc1.ParamByName('Line4').AsString;
Label5.Caption :=
  StoredProc1.ParamByName('Line5').AsString;
Label6.Caption :=
  StoredProc1.ParamByName('Line6').AsString;
```

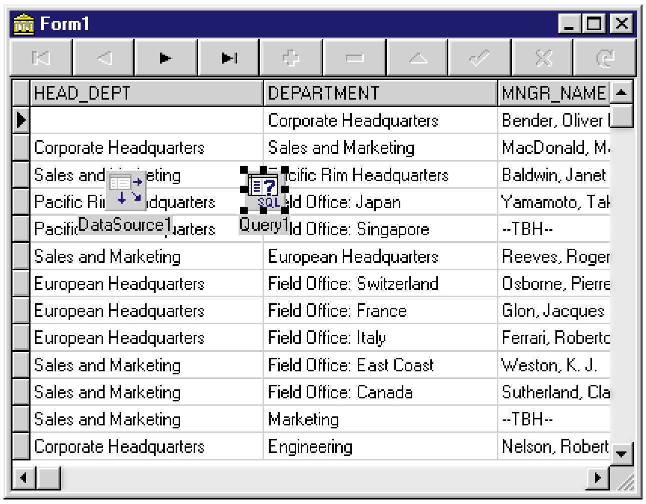
This stored procedure has seven parameters. *CUST_NO* is an input parameter; it must be assigned a value before executing the stored procedure. The parameters named *Line1*, *Line2*, etc., are output parameters, and are assigned a value as a result of the execution of the stored procedure.

11) Now, run the project. Enter the password `masterkey` when prompted. When the project is running, your screen should look something like Figure 5. Now navigate between the various records in the DBGrid. Notice that the mailing label represented by the Label components is updated each time you arrive on a new record.

Stored Procedure: Example 2

This next example demonstrates how to execute an InterBase stored procedure that returns a result set. As mentioned earlier, this involves the use of a Query component.

1) Create a new project. Add one DBNavigator, one DBGrid, one DataSource, and one Query component to the main form.
2) Set the *Align* property of the DBNavigator to *alTop*, and the *Align* property of the DBGrid to *alClient*. Set the *DataSource* property for both the DBNavigator and



HEAD_DEPT	DEPARTMENT	MNGR_NAME
	Corporate Headquarters	Bender, Oliver I
Corporate Headquarters	Sales and Marketing	MacDonald, M.
Sales and eting	cific Rim Headquarters	Baldwin, Janet
Pacific Ri dquarters	ld Office: Japan	Yamamoto, Tak
Pacifi arters	ld Office: Singapore	--TBH--
Sales and Marketing	European Headquarters	Reeves, Roger
European Headquarters	Field Office: Switzerland	Osborne, Pierre
European Headquarters	Field Office: France	Glon, Jacques
European Headquarters	Field Office: Italy	Ferrari, Roberto
Sales and Marketing	Field Office: East Coast	Weston, K. J.
Sales and Marketing	Field Office: Canada	Sutherland, Cla
Sales and Marketing	Marketing	--TBH--
Corporate Headquarters	Engineering	Nelson, Robert

Figure 6: The result set returned by a stored procedure on the Local InterBase Server is accessed using a Query component.

DBGrid to *DataSource1*.

3) Select *DataSource1* and set its *DataSet* property to *Query1*.
4) Now select *Query1* and set its *DatabaseName* property to *IBLOCAL* and its *SQL* property to `SELECT * FROM ORG_CHART`.
5) Finally, set the Query's *Active* property to True. This causes the SQL statement to execute, producing a call to the ORG_CHART stored procedure. Your main form should look like Figure 6. You can run the project; it will result in a display similar to the one you see at design time.

Conclusion

The DBDataSet components permit you to work with data stored in a database. The component you use for a particular application will usually depend on the type of database you're working with, and the performance and features you want to use. The next "DBNavigator" will take a look at the Database component. ∆

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.

New & Used

By Bill Todd

Wise Installation System 6.01

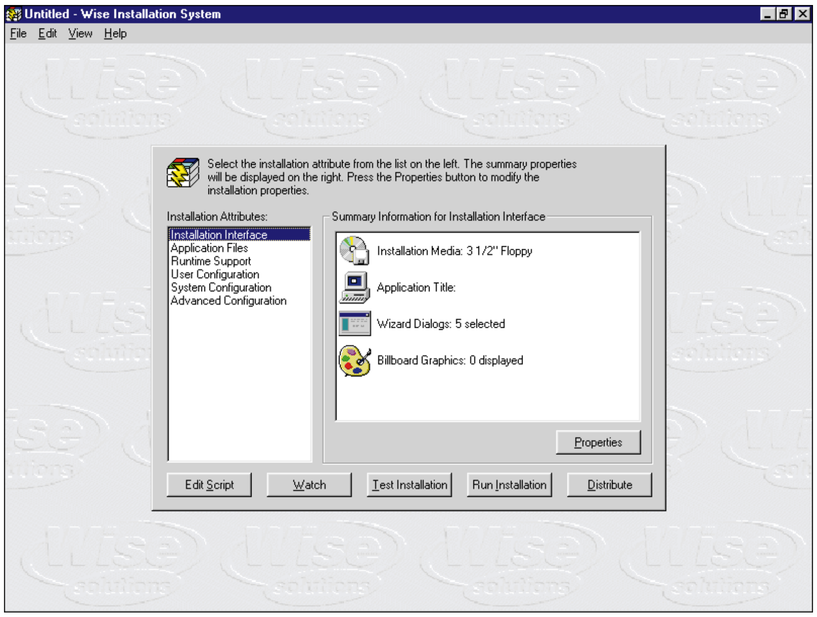
The Installation Program

If you need to install the Borland Database Engine (BDE) with your Delphi applications, Wise Installation System version 6.01 is the installation program you've been waiting for. Wise offers complete control of the BDE installation, including the directories in which the BDE and its configuration file will be installed, and every setting in the BDE configuration file.

The Installation Base

Wise Installation System is a script-based installation program that includes 16- and 32-bit versions, and contains an excellent wizard to build the script for you. With Wise, you have the best of both worlds. The wizard lets you create most installations quickly and easily. However, when you need to do something special, you can switch to the script editor with a single mouse click, and have total control.

When you start Wise, the opening screen of the installation wizard appears as shown in Figure 1. As you move through the installation attributes in the list box on the left, the icons and titles on the right change. These icons represent a page in a dialog box, giving you instant access to all of them, no matter which one you start with. You'll always begin by double-clicking **Installation Interface** to access the dialog box shown in Figure 2.



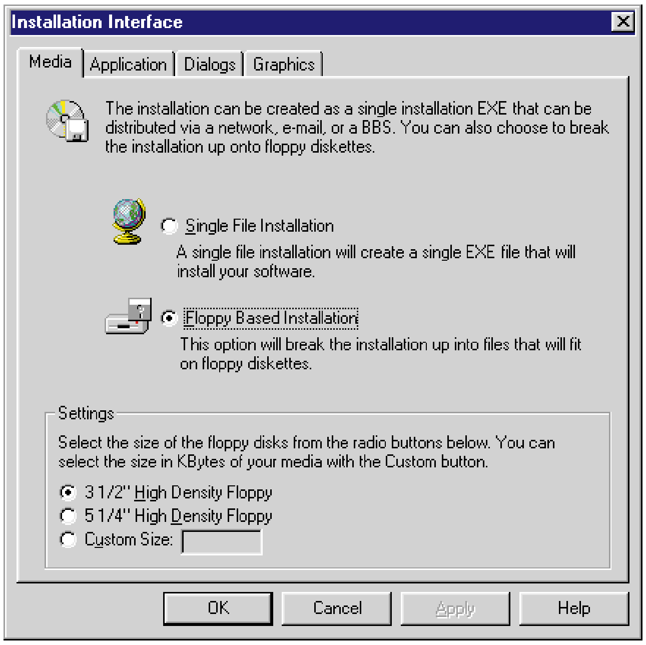
Select the installation attribute from the list on the left. The summary properties will be displayed on the right. Press the Properties button to modify the installation properties.

Installation Attributes:
- Installation Interface
- Application Files
- Runtime Support
- User Configuration
- System Configuration
- Advanced Configuration

Summary Information for Installation Interface
- Installation Media: 3 1/2" Floppy
- Application Title:
- Wizard Dialogs: 5 selected
- Billboard Graphics: 0 displayed

Figure 1: The opening screen of the installation wizard.

Wise supports both single-file and diskette-based installations. Single-file installation provides a single executable that contains the entire installation. This option is perfect for installations from a file server or CD-ROM, or to let users download the entire installation from the Internet or an intranet. If you select diskette installation, you have the option of specifying a custom size in addition to the standard 3 1/2-inch or 5 1/4-inch diskettes. One feature I particularly like is that Wise checks the current working directory before it prompts you to insert the next diskette during the installation process. This means that even if you deliver your installation on several diskettes, your users can copy the single file from each diskette to a directory on a file server, and run the installation from there.


Installation Interface

The installation can be created as a single installation EXE that can be distributed via a network, e-mail, or a BBS. You can also choose to break the installation up onto floppy diskettes.

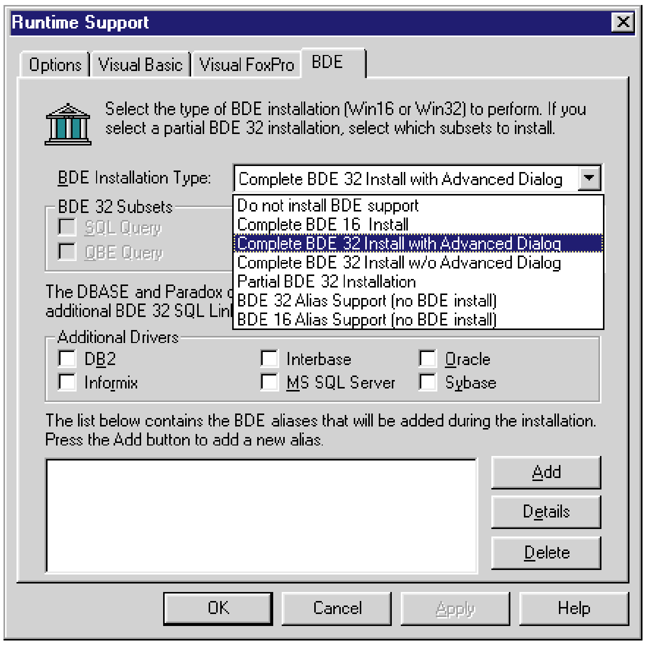
○ Single File Installation
A single file installation will create a single EXE file that will install your software.

[✓ Floppy Based Installation]
This option will break the installation up into files that will fit on floppy diskettes.

Settings
Select the size of the floppy disks from the radio buttons below. You can select the size in KBytes of your media with the Custom button.

[✓ 3 1/2" High Density Floppy]
○ 5 1/4" High Density Floppy
○ Custom Size:

Figure 2: The Installation Interface dialog box.


Runtime Support

Select the type of BDE installation (Win16 or Win32) to perform. If you select a partial BDE 32 installation, select which subsets to install.

BDE Installation Type: Complete BDE 32 Install with Advanced Dialog

- Do not install BDE support
- Complete BDE 16 Install
- Complete BDE 32 Install with Advanced Dialog
- Complete BDE 32 Install w/o Advanced Dialog
- Partial BDE 32 Installation
- BDE 32 Alias Support (no BDE install)
- BDE 16 Alias Support (no BDE install)

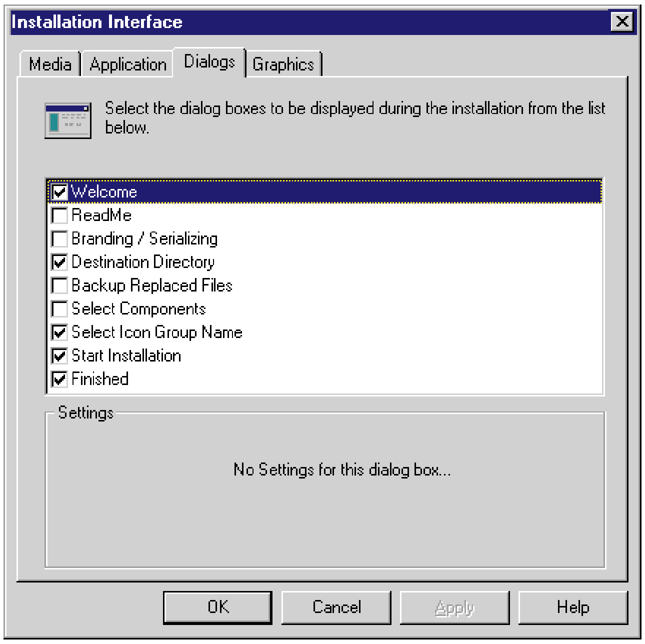
BDE 32 Subsets
☐ SQL Query
☐ QBE Query

The DBASE and Paradox d
additional BDE 32 SQL Lin

Additional Drivers
☐ DB2 ☐ Interbase ☐ Oracle
☐ Informix ☐ MS SQL Server ☐ Sybase

The list below contains the BDE aliases that will be added during the installation. Press the Add button to add a new alias.

Figure 4: The BDE options provided by the installation wizard.


Installation Interface

Select the dialog boxes to be displayed during the installation from the list below.

- ☑ Welcome
- ☐ ReadMe
- ☐ Branding / Serializing
- ☑ Destination Directory
- ☐ Backup Replaced Files
- ☐ Select Components
- ☑ Select Icon Group Name
- ☑ Start Installation
- ☑ Finished

Settings

No Settings for this dialog box...

Figure 3: The Dialogs tab in the Installation Interface dialog box.

Using the Application tab of the Installation Interface dialog box, you specify the name of your application and the directory in which to install it. You can have the installation directory automatically placed in the user's Program Files directory, or you can specify a complete path. Wise includes a variety of standard dialog boxes, and the Dialogs tab lets you choose which ones the users will see during the installation (see Figure 3). The options include:

- displaying a ReadMe file,
- prompting for the user's and company name,
- saving a backup copy of any files that are replaced,
- choosing the destination location, and
- selecting the components to be installed.

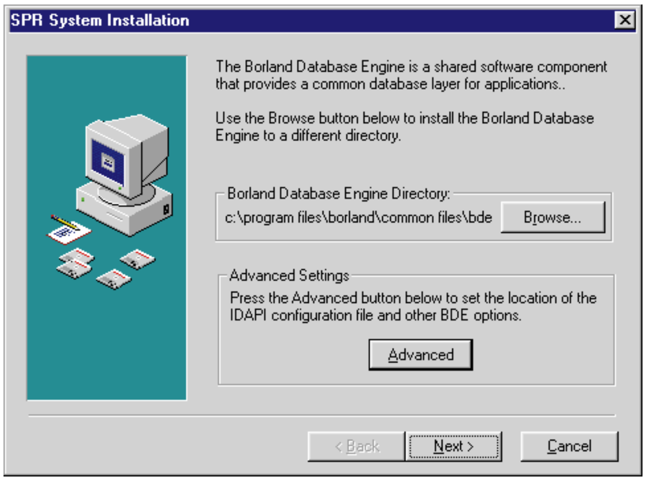
The option to omit the dialog box that lets you specify the installation location can be particularly handy if you're trying to enforce a consistent configuration on all machines. While the ability to select from the list of standard dialog boxes is nice, the real power lies in the dialog-box editor. Using this tool, you can modify any of the standard dialog boxes, or create your own, and add them to your installation.

File Management

Wise gives you complete control over the files you include in your application. You can create any number of file groups and name them anything you wish. You can also choose whether a file group will be installed by default or not. Your file groups appear in the Installation Components dialog box during installation. Those installed by default are check marked. You can change which file groups are selected by clicking them. Using file groups, you can easily create an installation using local tables where the empty database tables are not installed by default. This reduces the risk of someone overwriting an existing database when trying to install on a new workstation at a later time.

Wise Choices

Wise provides run-time installation support for Microsoft Visual Basic, Microsoft Visual FoxPro, and a host of ODBC drivers. However, for Delphi developers, it's the BDE support that makes Wise stand head-and-shoulders above its competition. Figure 4 shows the BDE options provided by the installation wizard. The **BDE Installation Type** drop-down menu shows the seven options. You can install the full 16- or 32-bit BDE, perform a partial 32-bit BDE installation, or add aliases to an existing 16- or 32-bit BDE installation. Check boxes let you select which, if any, SQL link drivers to include in your BDE installation.



SPR System Installation

The Borland Database Engine is a shared software component that provides a common database layer for applications..

Use the Browse button below to install the Borland Database Engine to a different directory.

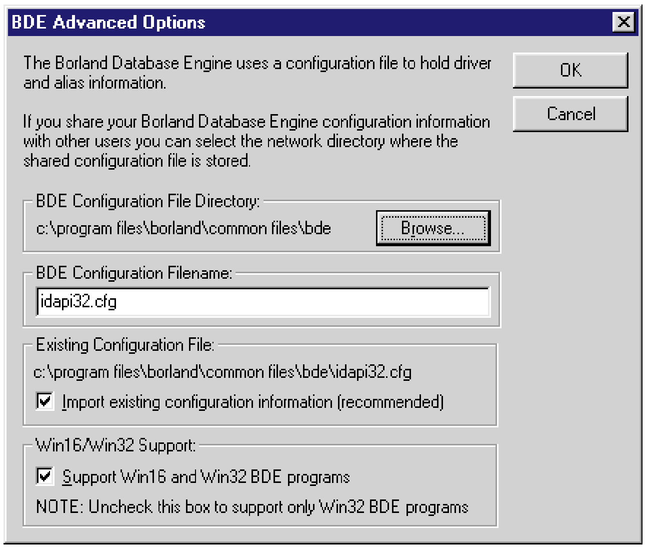
Borland Database Engine Directory:
c:\program files\borland\common files\bde

Advanced Settings
Press the Advanced button below to set the location of the IDAPI configuration file and other BDE options.

Figure 5: The BDE installation screen.

If you install the full 32-bit BDE, you have your choice of including the Advanced Dialog or not. Figure 5 shows the screen the end-user sees when installing an application that includes the BDE with the Advanced Dialog option. The **Browse** button for the BDE lets you choose where to install the BDE itself. This makes it possible for the user to install a single copy of the BDE to a shared directory on a file server so it can be shared by all users on the network.

Clicking the **Advanced** button displays the dialog box shown in Figure 6, which lets the user select the location for the BDE configuration file and its name. Users can also choose whether to import existing configuration file information, and whether to save the configuration file in 16-bit compatible format or not.



BDE Advanced Options

The Borland Database Engine uses a configuration file to hold driver and alias information.

If you share your Borland Database Engine configuration information with other users you can select the network directory where the shared configuration file is stored.

BDE Configuration File Directory:
c:\program files\borland\common files\bde

BDE Configuration Filename:
idapi32.cfg

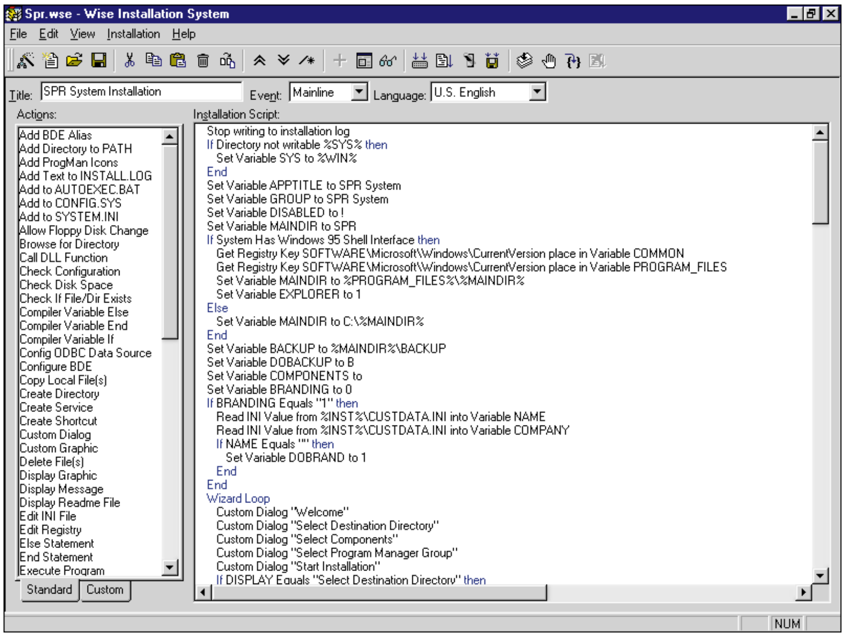
Existing Configuration File:
c:\program files\borland\common files\bde\idapi32.cfg
[✓] Import existing configuration information (recommended)

Win16/Win32 Support:
[✓] Support Win16 and Win32 BDE programs
NOTE: Uncheck this box to support only Win32 BDE programs

Figure 6: The BDE Advanced Options dialog box.

Wise supports all four possible BDE installation configurations:
1) placing the BDE and configuration file on each workstation,
2) placing the BDE on each workstation but letting all users share a single configuration file on the server,
3) placing the BDE and configuration file on the server, or
4) placing the configuration file on each workstation, and the BDE on the server.

INPRISE recommends putting the BDE on each workstation to provide the best performance. However, with large, high-performance networks, many organizations find that the ease of administration and upgrade installation they get by having a single shared BDE and configuration file more than compensates for any decrease in performance they experience. Regardless of the BDE configuration option you favor, Wise gives you the option to configure the BDE to best meet your needs. Note that you must save the BDE configuration file in 16-bit compatible format if all the users on a network will share a single configuration file. Therefore, you might want to use the dialog-box editor to remove these check boxes.



Spr.wse - Wise Installation System

Title: SPR System Installation
Event: Mainline
Language: U.S. English

Actions:
Add BDE Alias
Add Directory to PATH
Add ProgMan Icons
Add Text to INSTALL.LOG
Add to AUTOEXEC.BAT
Add to CONFIG.SYS
Add to SYSTEM.INI
Allow Floppy Disk Change
Browse for Directory
Call DLL Function
Check Configuration
Check Disk Space
Check If File/Dir Exists
Compiler Variable Else
Compiler Variable End
Compiler Variable If
Config ODBC Data Source
Configure BDE
Copy Local File(s)
Create Directory
Create Service
Create Shortcut
Custom Dialog
Custom Graphic
Delete File(s)
Display Graphic
Display Message
Display Readme File
Edit INI File
Edit Registry
Else Statement
End Statement
Execute Program

Installation Script:
```
Stop writing to installation log
If Directory not writable %SYS% then
  Set Variable SYS to %WIN%
End
Set Variable APPTITLE to SPR System
Set Variable GROUP to SPR System
Set Variable DISABLED to !
Set Variable MAINDIR to SPR
If System Has Windows 95 Shell Interface then
  Get Registry Key SOFTWARE\Microsoft\Windows\CurrentVersion place in Variable COMMON
  Get Registry Key SOFTWARE\Microsoft\Windows\CurrentVersion place in Variable PROGRAM_FILES
  Set Variable MAINDIR to %PROGRAM_FILES%\%MAINDIR%
  Set Variable EXPLORER to 1
Else
  Set Variable MAINDIR to C:\%MAINDIR%
End
Set Variable BACKUP to %MAINDIR%\BACKUP
Set Variable DOBACKUP to B
Set Variable COMPONENTS to
Set Variable BRANDING to 0
If BRANDING Equals "1" then
  Read INI Value from %INST%\CUSTDATA.INI into Variable NAME
  Read INI Value from %INST%\CUSTDATA.INI into Variable COMPANY
  If NAME Equals "" then
    Set Variable DOBRAND to 1
  End
End
Wizard Loop
  Custom Dialog "Welcome"
  Custom Dialog "Select Destination Directory"
  Custom Dialog "Select Components"
  Custom Dialog "Select Program Manager Group"
  Custom Dialog "Start Installation"
  If DISPLAY Equals "Select Destination Directory" then
```

Figure 7: The Wise script editor, which offers more options than the installation wizard.

Wise also lets you set any option in the BDE configuration file as part of your installation. For example, to set the Paradox driver's *NetDir* parameter to the BDENet directory under your main installation directory, just add the line:

```
\DRIVERS\PARADOX\INIT\NET DIR:
%MAINDIR%\BDENet
```

to the parameters list for any alias you're creating. The path in this example is the same path used by the BDE API function *DbiOpenCfgInfoList* to modify the BDE configuration file. You can find a complete list of these paths at http://www.inprise.com/devsupport/bde/bdeapiex in the example for *DbiOpenCfgInfoList*. This feature lets you say goodbye to writing separate programs that use complex BDE API calls to configure your BDE installation the way you want it.

If you need to go beyond the options the installation wizard provides, a single mouse click will take you to the Wise script editor shown in Figure 7. The script editor is powerful and easy to use. All the script commands are shown in the left-hand list box. To add a command to the script, drag and drop it where you want it. To set the options for the command, right click it, choose **Edit**, and a dialog box will appear that prompts you for any parameters or other information the script command requires. This approach gives you the power of a complete programming language without the need to learn the language and its syntax.

The scripting language is complete with a full set of looping and flow-control statements, as well as commands to do everything from creating program manager icons, registry entries, and directories, to reading and writing text files. Wise also includes a debugger for solving any problems with your scripts, as well as a test mode so you can see what your installation looks like without actually installing any files.

Wise Web Support

The Enterprise version of Wise also includes a patch file utility and a Web deployment option. The patch feature allows you to create patch files that will update one or more older versions of your program, or any of its files to a new version. Patch files are a great way to distribute updates to a program over the Internet or any low-speed connection because they are typically much smaller than the file being patched. Another advantage is that you can put patch files on a public Web site because they are useless to anyone who doesn't have the original program.

The WebDeploy system allows users to install software using files on a Web server. Only the files needed for the options the user selects are downloaded to the user's computer. This can save a substantial amount of file transfer time if your software offers many options. The one problem with WebDeploy is that users aren't left with a set of files they can reinstall from if necessary. It also means that the user could be left with a partially installed program if the connection to the Web is lost. For these reasons, the WebDeploy system seems suitable for a corporate intranet, but not for commercial software distribution over the Internet.

Conclusion

Wise Installation System 6.01 is an outstanding tool for installing Delphi applications. Nothing I've seen comes close to the complete support that Wise offers for installing and configuring the BDE. With the wizard, easy-to-use scripting language, dialog-box editor, and debugger, you can build for any Delphi application an installation that does everything you need. ∆

Bill Todd is President of The Database Group, Inc., a database consulting and development firm based near Phoenix. He is co-author of four database programming books, including *Delphi: A Developer's Guide* [M&T Books, 1995] and *Creating Paradox for Windows Applications* [New Riders Publishing, 1994]. He is a member of Team Borland, providing technical support on the Borland newsgroups, and has been a speaker at every Borland Developers Conference. He is also a Contributing Editor to *Delphi Informant*. He can be reached at BillTodd@compuserve.com or (602) 802-0178.

INFORMANT FACT FILE

Wise Installation System version 6.01 is the installation program you've been waiting for. Nothing comes close to the complete support that Wise offers for installing and configuring the BDE. With the wizard, easy-to-use scripting language, dialog-box editor, and debugger, you can build an installation that does everything you need for any Delphi application.

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

Phone: (734) 981-4970
Fax: (734) 981-9746
E-Mail: sales@wisesolutions.com
Web Site: http://www.wisesolutions.com
Price: US$299; Enterprise Edition, US$699

NEW & USED

By Alan C. Moore, Ph.D.

WPTools

An Everyday Library of Word Processing Components

Every day, it seems, our users expect more from our applications. In text editing or word processing, plain text just doesn't cut it anymore. INPRISE knew this and included the *TRichText* control beginning with Delphi 2. However, if you want to use rich text features in a 16-bit application, you're out of luck. And while the *TRichText* control in Delphi 2 and beyond is a big step up from *TMemo*, it can't help in a lot of specialized situations where you want more control over the appearance of text. If you're facing such a dilemma, the product we'll be examining in this review, WPTools 2.07 by Julian Ziersch of Munich, Germany, may be the answer.

Overview

WPTools consists of over 20 components (see Figure 1), support classes, and several other low-level classes (referred to as data structures). Built from the ground up, these components aren't dependent upon classes new to the 32-bit Delphi versions, and are less likely to be affected by changes that might occur in future versions of Delphi. They bring .RTF file support to Delphi 1 for 16-bit Windows applications (see Figure 2). The sidebar, "Rich Text Format — A Quick Refresher" on page 42, reviews RTF for those who may be unfamiliar with it.

First, we'll take a detailed look at the central component, *TWPRichText*. After that, we'll discuss the other components, beginning with the user interface components most closely associated with *TWPRichText*. We'll conclude by looking at some of the newer features, and by discussing the library's strengths and weaknesses.

TWPRichText: A Superior RTF Component

Before discussing *TWPRichText*, let's take a look behind the scenes. The *WPReader* support class takes care of all the gory details of reading and interpreting the various identifiers, tags, and special characters. During its initialization, this unit reads 170 such items (including bitmap identifiers) into its internal storage.

The main component, *TWPRichText*, is derived from *TWPCustomRichText*. The latter is a subclass of *TWPCustomRtfEdit*, which is the base class. It wraps a *Memo* object, which includes most of the RTF functionality, but doesn't have its own handle. And, like its Delphi cousin, *TRichEdit*, it's similar in behavior to *TMemo*. But rather than simply working with simple ASCII text, it takes care of the basic tasks of reading, writing, and displaying rich text like its VCL counterpart.

TWPRichText supports two resolution modes: device dependent, which is fast but not WYSIWYG; and device independent, which provides true WYSIWYG viewing and fast zooming. There are also two memory allocation modes: rich and plain text. The latter consumes less memory.

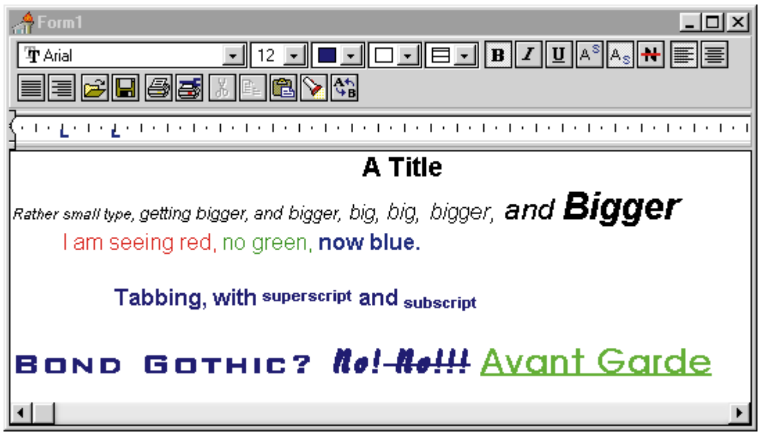
TWPRichText's large number of properties and methods provide the starting point for building a fully functional word processor, or adding sophisticated text formatting to any application. Several properties give you control over text formatting. The current

Component	Description
TWPRichText	Edits formatted text: alignment, paragraph spacing, borders, font selection, and colors.
TDBWPRichText	Stores text in a database in Memo or BLOB fields. Can switch between RTF or normal ANSI formats.
TWPRtfStorage	Stores formatted text and assigns it to any visual RTF component. Usually used at design time.
TWPRichTextLabel	Displays formatted static text that can't be scrolled or edited. The text can be transparent.
TWPRuler	A ruler component managed by *TWPRichText*. Units can be set to either centimeter or inches, even at run time.
TWPToolBar	Controls one or more *TWPRichText* or *TDBWPRichText* components with pre-defined speed buttons and list boxes.
TWPToolPanel	Can handle *TWPComboBox* and *TWPToolButton*, and enables automatic alignment of controls.
TWPToolCtrl	Helps you use third-party toolbar components with *TWPToolButton* and *TWPComboBox*.
TWPComboBox	Used with *TWPToolPanel*, it adds functionality to control *TWPRichText*.
TWPToolButton	Similar to *TSpeedButton*. Allows you to add functionality to control *TWPRichText*.
TWPStatusBar	Can be used in most projects. It uses strings that can be accessed from the program. You can also use a gauge.
TWPParagraphBorderDlg	Displays a dialog box to set up or edit paragraph borders.
TWPParagraphPropDlg	Displays a dialog box to set up the paragraph indentation and spacing.
TWPWallPaper	Operates similarly to *TImage*, but it will tile the picture.
TWPSpellCheckDlg	Containing source code, implements spell-checking for *TWPRichText* or *TDBWPRichText*. (WPTools doesn't have its own dictionary, so a third-party spell-checker is required.)
TWPEdit	Based on *TWPCustomRichText*, provides single-line, RTF-formatted text.
TWPPreview	Displays the contents of an attached *TWPRichText* component in multiple pages, and with rows and columns.
TWPQuickPrint	A printing component. To create reports, place a *TQRBand* object and *TWPDBQuickPrint*.
TWPDBQuickPrint	Similar to *TWPQuickPrint*; automatically retrieves data from *TDataSource* (e.g. *TDBMemo*).
TWPAltStatusBar	A basic status bar suitable for most projects.
TWPPreView	A preview control that can display multiple pages in multiple rows and columns.
TWPPreviewDlg	A preview dialog box that provides easy access to *TWPPreview*.
TWPVertRuler	A component that displays a vertical ruler (no scrolling yet).

Figure 1: Current WPTools components.

set of writing characteristics is stored in the property *Attr*, and can be changed with the procedure, *ChangeAttr*. The latter has two parameters: a *TAttr* structure (like *Attr* itself) and a set containing the attributes themselves, *TWhatToChange*. *FastResetAttr* resets the attributes to their default values. I've found this structure very useful.

Text attributes are defined in the *TPersistent* subclass, *TCharacterAttr*. They include the state of each of the following: bold, italic, underline, strikeout, superscript, and subscript; the state for each can be *tsIgnore*, *tsTrue*, or *tsFalse*. *TCharacterAttr* can be used to change the way special text (e.g. hyperlinks that can be underlined) is displayed. It can also display text with a certain background. The new version adds a new property, *CurrAttr*, making it easier to change text attributes, paragraph formatting, tabs, and borders. In addition to hiding text, you can change the text, background, and underline colors, and indicate if they should be used or ignored. You can have a background image or even make the background transparent, and can load the image from a file, making it possible to load background images with HTML files.



A Title

Rather small type, getting bigger, and bigger, big, big, bigger, and Bigger

I am seeing red, no green, now blue.

Tabbing, with superscript and subscript

BOND GOTHIC? No! No!!! Avant Garde

Figure 2: WPTools provides RTF and HTML support to Delphi 1 for 16-bit Windows applications.

Structuring Documents

We generally organize a document's structure in lines, paragraphs, and pages. WPTools provides much support for this type of organization, particularly paragraphs. You can retrieve the number of the current line and load it into a string. In working with paragraphs, you can set or change their attributes (including indentation), read all the text into a buffer, change their ordering by moving paragraphs, set their alignment (left, right, center, block), and set their border properties, margins, and spacing.

Group	Function	Action
sel_ActionIcons	SelExit	Exit application
sel_ActionIcons	SelNew	New file in editor
sel_ActionIcons	SelOpen	Open existing file
sel_ActionIcons	SelSave	Save file
sel_ActionIcons	SelClose	Close file
sel_ActionIcons	SelPrint	Print file
sel_ActionIcons	SelPrintSetup	Set up printer
sel_EditIcons	SelCopy	Copy selection
sel_EditIcons	SelCut	Cut selection
sel_EditIcons	SelPaste	Paste selection
sel_EditIcons	SelSelAll	Select all
sel_EditIcons	SelHideSel	Hide selection
sel_EditIcons	SelFind	Find selection
sel_EditIcons	SelReplace	Replace selection
sel_EditIcons	SelSpellCheck	Execute spell check (event)
sel_ListBoxes	SelFontName	Font-name list box
sel_ListBoxes	SelFontSize	Font-size list box
sel_ListBoxes	SelFontColor	Font-color list box
sel_ListBoxes	SelBackgroundColor	Background-color list box
sel_ListBoxes	SelParColor	Paragraph-color list box
sel_ListBoxes	SelStyle	User-defined list box

Figure 3: Some groups of buttons available in *TWPToolBar*.

If you're saving to the HTML format, you can add further structuring. For example, you can create an automatic list with numbers or bullets, and indicate page breaks, retrieve the current page number, get the starting line of a page, and get the starting paragraph line of the page.

The *BackgroundScrolling* property lets you optimize the scroll speed by scrolling the background image with the text. You can control the visual appearance of bookmark, header, HTML, and protected text, as well as the insertion point. Many properties and methods for navigating text are included. For example, the *GetPosition* method retrieves the current absolute cursor position from the start of the text.

We've examined this central component. Now let's take a look at the supporting cast.

Basic Interface Components

The components often used with *TWPRichText* include *TWPToolBar*, *TWPRuler*, *TWPVertRuler* (new to this ver-

Rich Text Format — A Quick Refresher

While most Delphi developers are aware of what Rich Text Format (RTF) means, a few may not be. Just in case, here's a quick refresher.

In the early days of personal computing, there was just ASCII — plain, unformatted text. Even the most sophisticated of today's word processing programs, such as Microsoft Word and WordPerfect, still allow users to save their work in this format.

However, application users are demanding. As word processors developed, each strove to provide the maximum in text formatting options, including the ability to change font faces, font sizes, and other characteristics. It was important to be able to do this at any point in a document — even in the middle of a line.

Each word processor developed its own proprietary format to include formatting information along with the characters themselves. But there needed to be a means of communicating not just ASCII characters between different word processing programs, but also common formatting options. Thus, RTF was born. The following line is from a rich text document:

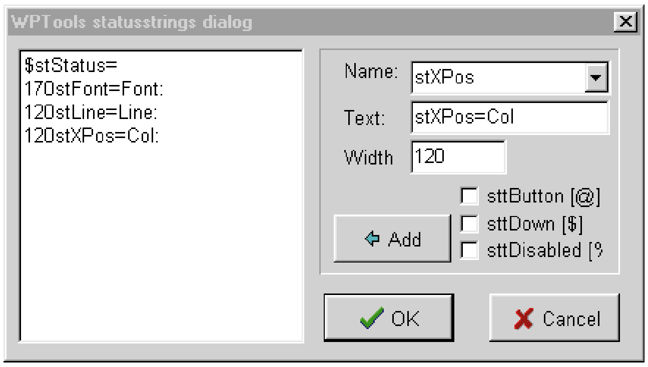
RTF normal; **RTF Bold**; *RTF Italic*; ***RTF Bold and Italic***; and <u>RTF underlined</u>.

When viewed in text format (and after deleting several lines of preliminary information on fonts, colors, and style characteristics), it looks like this:

```
\deflang1033\pard\plain\f2\fs20 RTF normal; \plain\f2\fs20\b RTF Bold;\plain\f2\fs20 \plain\f2\fs20\i RTF
Italic;\plain\f2\fs20 \plain\f2\fs20\b\i RTF Bold and Italic;\plain\f2\fs20 and \plain\f2\fs20\ul RTF underlined.\plain\f2\fs20
\par }
```

The RTF codes shown above represent a small sample of this rich (forgive the pun) language. .RTF files are also used as the main source file for Windows Help file systems. Unless you're writing your own Help-authoring program or RTF component, you needn't be concerned with the details of the RTF language itself — only about how you can use rich text to enhance your applications.

— Alan C. Moore, Ph.D.



$stStatus=
170stFont=Font:
120stLine=Line:
120stXPos=Col:

Name: stXPos
Text: stXPos=Col
Width 120
sttButton [@]
sttDown [$]
sttDisabled [?

Figure 4: The statusstrings dialog box makes it easy to set up a status bar.

sion), and *TWPStatusBar*. *TWPToolBar* comes with a series of built-in tool buttons that enable many common operations, such as opening or saving files, changing font type, size, and characteristics — such as italic or bold face — and changing the alignment. With the *TWPToolCtrl* component, you can use third-party toolbar components with *TWPToolButton*.

Linked to *TWPRichText* as one of its properties, *TWPToolBar* provides a great deal of built-in functionality. Figure 3 shows the most important groups of buttons and their functions. The *sel_ActionIcons* group triggers many applications and file-manipulation actions; the *sel_EditIcons* group provides easy access to the various text-editing options; and the *sel_ListBoxes* group makes useful list boxes available. In many cases, you enable their functionality by simply checking the items in each of these categories. In a few cases, you need to write a minimal amount of code to execute the event. The *OnIconSelection* event allows you to intercept clicks on the tool buttons.

In addition to these groups, there are the *sel_StatusIcons*, which toggle various editing modes, including *SelNormal*, *SelBold*, *SelItalic*, *SelUnder*, *SelHyperLink*, *SelStrikeOut*, *SelSuper*, *SelSub*, *SelLeft*, *SelHidden*, *SelRTFCode*, *SelLeft*, *SelRight*, *SelBlock*, and *SelCenter*. With WPTools' strong database support (which we'll be discussing), there are a number of database-related buttons that perform database navigating and selecting operations: *SelToStart*, *SelNext*, *SelPrev*, *SelToEnd*, *SelEdit*, *SelAdd*, *SelDel*, *SelCancel*, and *SelPost*. *TWPStatusBar* is a very flexible component that can be used in many types of projects. It generally uses a set of strings that can be easily manipulated from a program; it can also use a gauge to show the ongoing status of an operation.

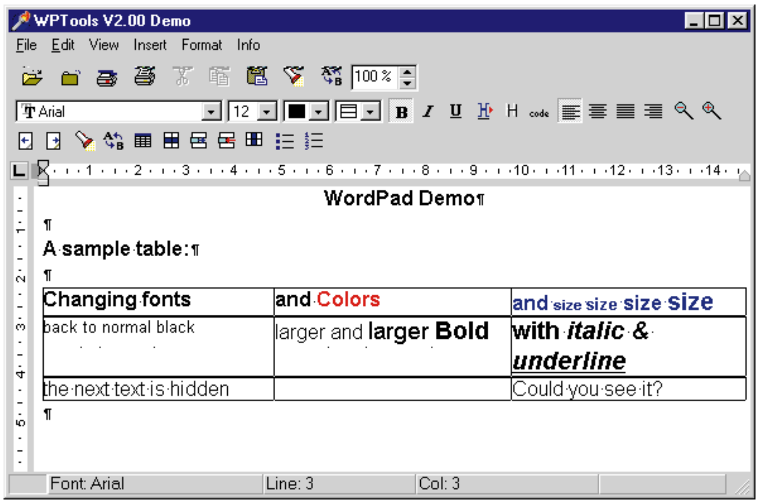
While you need to write a bit of code to implement many of the status-bar messages, it's not difficult. By clicking *TWPAltStatusBar*'s *String* property in the Object Inspector, you bring up the statusstrings dialog box (see Figure 4). Here, you can select from a number of pre-defined status items to set their properties. After that, you need to define event handlers and you're ready to roll. To set up the status bar shown in Figure 5, I simply wrote the following three methods, adding just one line of code to each:

```
procedure TWordPadForm.WPRichText1CharacterAttrChange(
  Sender: TObject; Attribute: TWPSetModeControl);
begin
  WPAltStatusBar1.SetString(stFont,'Font: ' +
    WPRichText1.CurrAttr.FontName);
end;

procedure TWordPadForm.WPRichText1XPosChanged(
  Sender: TObject);
begin
  WPAltStatusBar1.SetString(stXPos,'Col: ' +
    IntToStr(WPRichText1.CPColNr+1));
end;

procedure TWordPadForm.WPRichText1YPosChanged(
  Sender: TObject);
begin
  WPAltStatusBar1.SetString(stLine, 'Line: ' +
    IntToStr(WPRichText1.GetLineNumber+1));
end;
```

WPTools includes two rulers — horizontal and vertical — that provide strong support for building desktop publishing applications. *TWPRichText* provides built-in table support. These features, and more, are shown in Figure 5.



WordPad Demo¶

¶

A sample table:¶

¶

Changing fonts	and Colors	and size size size size
back to normal black	larger and larger Bold	with italic & underline
the next text is hidden		Could you see it?

¶

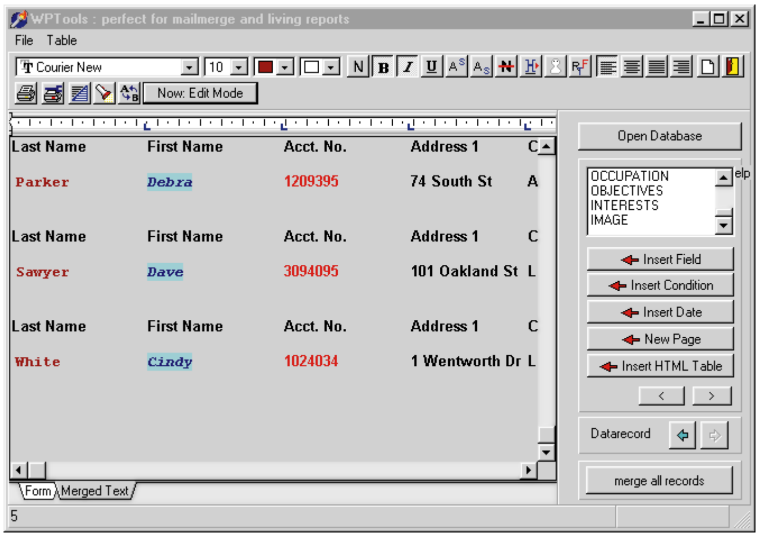
Font: Arial Line: 3 Col: 3

Figure 5: WPTools' interface components — toolbar, two rulers, and status bar — allow you to build feature-rich applications.

Up to this point, we've concentrated on basic word processing tools. But there is quite a bit more to this library.

Beyond Word Processing

Besides supporting word processing, WPTools provides a data-aware version of its main component; it also provides advanced graphics- and object-manipulation capabilities. Figure 6 shows an application that demonstrates WPTools' database support. Note how much control you have over the formatting of individual fields. This sample application also includes database opening and navigation controls, and features WPTools' useful merging capabilities.


WPTools : perfect for mailmerge and living reports

Now: Edit Mode

Last Name First Name Acct. No. Address 1
Parker Debra 1209395 74 South St

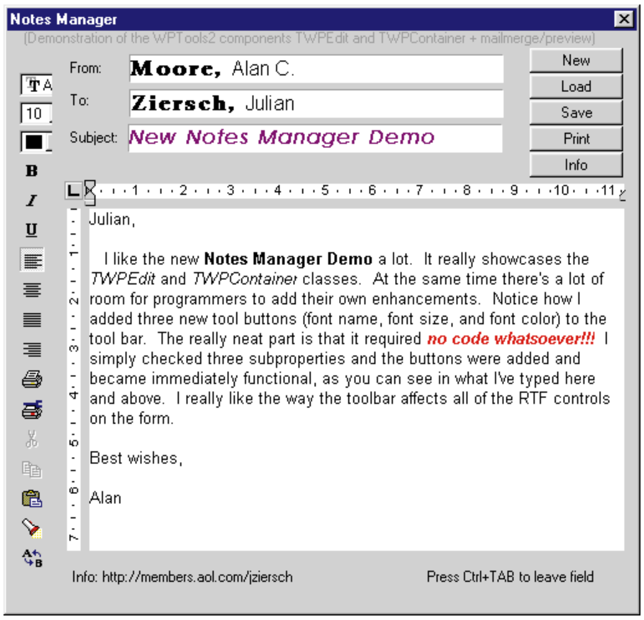
Last Name First Name Acct. No. Address 1
Sawyer Dave 3094095 101 Oakland St

Last Name First Name Acct. No. Address 1
White Cindy 1024034 1 Wentworth Dr

Figure 6: The database support allows you to format individual fields.

This library also includes built-in drag-and-drop support. When enabling drag-and-drop, you can choose to either keep or discard RTF formatting. Of course, you could hard-code this into your application, or give users the choice. If you need to create an application with single-line RTF edit controls (rather than memo controls), WPTools' *TWPEdit* supports this, as shown in Figure 7. Note that the formatting buttons at the left control the three RTF edit controls at the top, as well as the RTF memo control at the bottom.

Finally, WPTools provides support for embedding graphics and other objects directly in an .RTF document, as shown in Figure 8. In addition to dropping text from one window to another, you can also program WPTools to drag controls, such as *TPanel* or *TButton*, to the memo. The text is retrieved using the *GetTextBuf* method, which exists in all descendants of *TControl*. This sample application shows quite a few embeddable objects, but there's really no limit. You can save the document with all the embedded objects. When you reload the document, all the objects appear in the proper locations within the document.


Notes Manager

(Demonstration of the WPTools2 components TWPEdit and TWPContainer + mailmerge/preview)

From: **Moore,** Alan C.

To: **Ziersch,** Julian

Subject: *New Notes Manager Demo*

Julian,

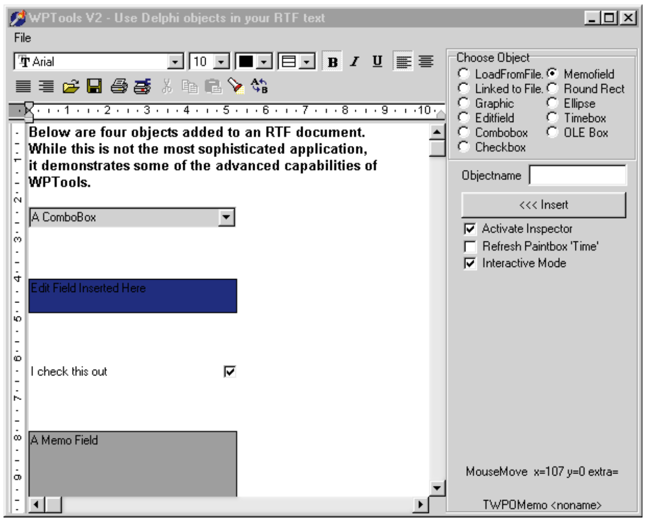
I like the new **Notes Manager Demo** a lot. It really showcases the *TWPEdit* and *TWPContainer* classes. At the same time there's a lot of room for programmers to add their own enhancements. Notice how I added three new tool buttons (font name, font size, and font color) to the tool bar. The really neat part is that it required ***no code whatsoever!!!*** I simply checked three subproperties and the buttons were added and became immediately functional, as you can see in what I've typed here and above. I really like the way the toolbar affects all of the RTF controls on the form.

Best wishes,

Alan

Info: http://members.aol.com/jziersch Press Ctrl+TAB to leave field

Figure 7: WPTools' *TWPEdit* supports single-line RTF editing.


WPTools V2 - Use Delphi objects in your RTF text

Below are four objects added to an RTF document. While this is not the most sophisticated application, it demonstrates some of the advanced capabilities of WPTools.

A ComboBox

Edit Field Inserted Here

I check this out ☑

A Memo Field

Choose Object
- ○ LoadFromFile
- ○ Linked to File
- ○ Graphic
- ○ Editfield
- ○ Combobox
- ○ Checkbox
- ● Memofield
- ○ Round Rect
- ○ Ellipse
- ○ Timebox
- ○ OLE Box

Objectname

<<< Insert

- ☑ Activate Inspector
- ☐ Refresh Paintbox 'Time'
- ☑ Interactive Mode

MouseMove x=107 y=0 extra=

TWPOMemo <noname>

Figure 8: You can embed, save, and load graphics or other objects directly in an .RTF document.

Conclusion

On the whole, I feel this is an excellent library. I've seen considerable improvement in the few short months I've been working with it. The new version includes significant enhancements, such as supporting headers and footers. It also has enhanced support for tables, including inserting rows, changing cell and column widths, and marking blocks of the table.

With WPTools' support for embedded objects, you can add support for third-party image libraries. There's also strong support for QuickReport versions 1 and 2 (two example programs), third-party toolbars, and spell-checking components. Because you have access to the vast majority of the source code (the code for the RTF engine is available separately), you can add additional support for other spell-checkers or reporting tools.

WPTools isn't without drawbacks, but I consider them minor. While the documentation has improved considerably

INFORMANT FACT FILE

WPTools from Julian Ziersch of Munich, Germany is a superb collection of Rich Text Format (RTF) components. Because it includes its own RTF engine, it gives developers a great deal of control over formatted text. In creating word processing applications, it supports standard paragraph properties (indentation, spacing, alignment, margins, and tabs), character attributes (bold, italic, superscript, subscript, color, etc.), and support for hyperlinks, protected text, hidden text, headers, footers, bookmarks, and mail-merge. Beyond its strictly RTF features, it allows the incorporation of graphics and Delphi objects in .RTF documents.

Julian Ziersch Software
Maximilianstrasse 25
80539 Munich, Germany

E-Mail: wptools@compuserve.com
Web Site: http://www.wptools.com
Price: WPTools 2.07, US$178; upgrade for owners of version 1.x, US$80.

since the first version, there are still some English errors here and there (the author is German, and while his command of English is excellent, the documentation could benefit from thorough editing). I also hope that a future version would include printed documentation. The final drawback comes from the memory-intensive nature of working with .RTF files. Screen updates and other tasks seemed a bit slow at times. Based on the changes I've seen in the past several months, I expect all of these factors to improve in future versions.

I'm very pleased with this library's design and features, and particularly with its creator's commitment to improving it. If you need RTF processing in your applications, I think you will be satisfied with WPTools. ∆

Alan Moore is a Professor of Music at Kentucky State University, specializing in music composition and music theory. He has been developing education-related applications with the Borland languages for more than 10 years. He has published a number of articles in various technical journals. Using Delphi, he specializes in writing custom components and implementing multimedia capabilities in applications, particularly sound and music. You can reach Alan on the Internet at acmdoc@aol.com.

table of CONTENTS

So Many Standards

Or, TLAs A Go Go

The pace of technology has been dizzying. Trying to keep up has given a new techno-meaning to the phrase "Keeping up with the Joneses." Just take a look at some of the possibilities: COM, DCOM, CORBA, MIDAS, MTS, MSMQ, IIS, ISAPI, HTTP, FTP, ASP, DDE, OLE, ODBC, OLEDB, ADO, VBA, and RDS.

I didn't even mention beta products, or other acronyms found throughout the IT landscape. For example: COM+, DNA, SMS, MFC, ATL, CDO, SAP, JFC, JDBC, and WFC.

You're doing well if you know what all these acronyms stand for. If you use them, you're doing even better!

When developers first moved to Windows, the learning curve was monumental. Remember your first foray into the Windows SDK while learning a new paradigm known as event-driven programming? Then, just as we got comfortable with *that* SDK and began to shorten *that* learning curve, we received the Win32 SDK, and a new technology revolution sprang to life. Learning curves have been steepened dramatically once again.

The question that begs to be asked is: "How has this advance in technology helped programmers in general and Delphi programmers specifically?"

Using new technology to solve problems that were previously time-consuming, or even impossible, is the best use for emerging technology. To do this successfully, you must have a firm grasp of technology and constantly stay on top of it to separate what will truly work for you — and what is just hype.

On the other hand, using new technology for technology's sake usually merits a red flag in any development effort. If you find you must incorporate technology into your application to be the first to market, expect that your schedule will not be as smooth as one using a tried and true approach. The benefits of incorporating leading-edge technology into your application will bring great rewards when you are the one who has a stronghold on the market. This also makes those bumps along the development path seem inconsequential in hindsight.

Technology changes — rapidly. That's a fact no one can dispute. You must be prepared to deal with change, even if you aren't one that likes to be on the forefront of technology. You can wait for the dust to settle and let the marketplace declare the winner, but you're then faced with the same learning curve. And if your competition is already using that technology, they will have an enormous advantage because they have already climbed that learning curve.

And speaking of the Joneses, I'm reminded of the guardian of the Holy Grail in the third Indiana Jones movie. In this movie, he guards the chalice, waiting for someone to come and drink from the cup. If they choose the true Holy Grail from a wall full of imposters, they will be rewarded with eternal life. If they choose the wrong cup, they face instant and certain death. While Indiana Jones is scouring the wall looking for the Holy Grail, the guardian says: "Choose wisely."

Similarly, if you drink from the wrong technology cup, you are certainly doomed. At least until you can make reparations with management and release your product.

The moral of the story? There are myriad choices out there — all of which seem promising. Knowing your business requirements, the skill set of your development team, and your customer base will certainly help you decide on a set of technologies that will best help you deliver a quality application. Once you've done your homework and can justify your position, you can certainly say you chose wisely. ∆

— Dan Miser

Dan Miser is a Design Architect for Stratagem, a consulting company in Milwaukee. He has been a Borland Certified Client/Server Developer since 1996, and is a frequent contributor to Delphi Informant. *You can contact him at http://www.execpc.com/~dmiser.*

table of CONTENTS

RAD Publishers and Authors: Delphi on the Web

In the April, 1998 issue of *Delphi Informant*, I discussed various vendor-related Delphi Web sites. This month, I'll examine sites related to Delphi publishers and authors. Vendors and publishers have one thing in common: They're in business. And like the vendor sites we examined earlier, the publisher and author sites provide valuable resources.

Three top programming publishers. For many years, the Cobb Group has been a leading publisher of software-related magazines, including the *Delphi Developer's Journal* (DDJ). Its Web site contains material related to just about every aspect of programming. For Delphi programmers, there are many downloadable files, some electronic articles, and information related to DDJ. The site provides a gateway to several Delphi list servers and a message board. I was disappointed, however, that not all of its lists were included — only the main Delphi thread I helped moderate.

Like Cobb, the Coriolis Group provides information on many development environments. A major producer of developer books and the eclectic journal, *Visual Developer Magazine* (with Ray Konopka's Delphi column), the VDM section of its site features many free downloads, including back issues in electronic form. Be sure to read Jeff Duntemann's excellent paper on "The Virtual Encyclopedia Initiative," a project intended to provide a solution to the increasingly difficult task of finding information on the Web.

Somewhat newer, the Informant Communications Group (ICG), publisher of this magazine, has certainly made its mark. As you would expect, there are sample articles, and the very first issue is in electronic format. The download section is among the best of the sites I've visited, rivaling some of the major Delphi sites. Not only can you download the source code for all the articles that have appeared in *Delphi Informant*, you can also download many third-party tools. Finally, ICG has provided a great service to the Delphi community, hosting Robert Vivrette's *Unofficial Newsletter of Delphi Users* (*UNDU*). If you've never read *UNDU*, you're missing out on a lot of free information.

Other Delphi magazines. No doubt many of you are familiar with *The Delphi Magazine*. Published in the United Kingdom, it includes articles by leading European authors. Its Web site includes sample articles, one entire issue, and files that accompany the various issues. The links to Delphi-related sites are especially good, and include some unusual ones I haven't seen anywhere else.

Delphi Developer, published by Pinnacle Press, also has a good Web site. I found its tips section particularly interesting and worthwhile. While they provide a sample of one of their issues, their downloads are restricted to subscribers only. What a shame.

Major authors. In April's issue, I made reference to Ray Konopka, Delphi author and tool producer. Because I'm discussing Delphi publishers, I'd like to mention the sites of three other authors: Marco Cantù, Ray Lischner, and Bob Swart. Cantù's site (http://www.marcocantu.com) is excellent, containing free downloads from his recent books, great links, book reviews, and other interesting material, such as opinion essays. Be sure to read his piece on INPRISE.

Ray Lischner's Web site (http://www.tempest-sw.com) contains useful links and downloads, including a Delphi 3 update of the *Secrets of Delphi 2* [Waite Group Press, 1996], and a Delphi 4 update for *Hidden Paths of Delphi 3* [Informant Press, 1997]. For a nice diversion, jump to the "Have Fun with the Spammers" page, and follow the links for a while.

In discussing authors' sites I would be remiss if I did not mention Dr. Bob's (http://www.drbob42.com). Bob Swart, one of the more popular presenters at INPRISE conferences, has a site full of downloads, book and tool reviews, great links, articles, and more.

Rating the sites. My summary evaluation of the publisher sites is given in the accompanying table. I recommend visiting them all, as well as the author sites. This winter I plan to return to this theme, exploring sites that are "off the beaten path." Some of them will be specifically related to Delphi, while others will feature general programming information of interest to Delphi developers. If you know of any that fit this description, please send them to me at acmdoc@aol.com. ∆

— 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 via e-mail at acmdoc@aol.com.

Web Site (http://...)	Article Downloads	Technical Papers	Communication Services	Samples	Links
Delphi Developer's Journal www.cobb.com/ddj	Excellent	Very good	Excellent	Very good	Limited
Visual Developer Magazine www.coriolis.com	Excellent	Good	None (coming soon)	Excellent	None
The Delphi Magazine www.itecuk.com/delmag/index.htm	Excellent	Good	Good	Good	Excellent
Delphi Developer www.pinpub.com/delphi/home.htm	Limited to subscribers	Very good	None	Good	Good
Delphi Informant www.informant.com	Excellent	Very good	None	Very good	Limited