

An Evaluation of
Rapid Application Development Tools
for Windows:
A Comparison Between
Delphi and Visual Basic

Table Of Contents

Introduction.....	
Windows Visual Development.....	
Performance.....	5
Benchmarks.....	5
Benchmark Results.....	8
Rapid Application Development (RAD).....	9
Controls.....	10
Control Icon Array.....	10
Templates and Experts.....	11
Object Placement.....	11
Property Lists.....	11
Code Window.....	11
Debugging And Object Inspecting.....	12
Component Reuse.....	13
Shared Event Functions.....	13
Reusable Functions and Libraries.....	13
Components.....	14
Programming Language.....	14
OOP Design Methodology.....	15
Database Scalability.....	16
Moving Up to Delphi.....	18
Conclusion.....	18

Introduction

With the growing demands on Windows applications developers to create increasingly complex applications in less time, the evolution of Rapid Application Development (RAD) tools has become a crucial focus of the development community. First-generation RAD tools for Windows included application frameworks (such as OWL and MFC), DLL-based class libraries, and VBXs (Visual Basic custom controls).

The release of Borland's Delphi heralds a new generation of RAD tools that combines the power of traditional 3GL compilers with the ease of use and development speed of a 4GL environment. This white paper will contrast the approach of the new technology used by Delphi with that of Microsoft's Visual Basic.

Windows Visual Development

The earliest methodology for Windows applications programming was to code directly using calls to the Windows API. The API provided a crude mechanism for creating such items as menus and windows, leaving the developer with an enormous coding task even when creating a rudimentary application. These early programs were typically created with C compilers that were equally crude by today's standards. As a result, the expectations for what could reasonably be achieved by professional developers was severely limited by schedule and performance constraints. Early applications were also, for the most part, independent. In other words, they were generally not reliant on each other and did not share data or invoke other applications.

Appreciating the underlying potential of the Windows environment, the development community demanded better tools, to enhance their productivity and facilitate the creation of more sophisticated applications. As Windows evolved, inter-application communications using DDE and OLE was introduced, then more powerful application frameworks products such as Borland's OWL and Microsoft's MFC appeared, as well as third-party products such as zApp from Zinc and Island Systems' object-Menu.

These libraries were used to encapsulate the most common functions of Windows applications, as well as to leave room for expansion and customization by developers. Thus, a programmer could quickly create a window with a certain border style, make it modal and add a "Close" button, then invoke it with a single call. Further, the advent of C++ compilers for Windows allowed experienced developers to exploit the power of object-oriented technology. Developers now had the means to create complex applications within acceptable schedules.

Object-oriented languages allowed developers to create classes and override specific virtual functions, providing a direct path to building custom libraries. This generation of tools still had two major shortcomings:

1. Although productivity had been significantly increased, schedules for complex applications development were still quite lengthy. For example, the common development scenario would proceed as follows:
 - a) some sample screens would be created in a prototype environment or a resource tool;
 - b) marketing would critique the screen design and modifications would be made;
 - c) the “final” screens would be integrated with code to complete the application;
 - d) any changes that were requested often became complicated and painful, since the code and the screens were so closely linked.
2. The expertise required made Windows programming the sole domain of the experienced developer. In other words, the extensive needs of the corporate community simply could not be met due to the requirement for significant programmer expertise.

Thus evolved the next stage of Windows development, characterized by 4GL visual design environments such as Microsoft Visual Basic and high-end products such as Powersoft’s PowerBuilder. These environments provided a major step forward in user-friendly development, with high-level, reusable components called controls which introduced the concept of a “building block” approach to software development. Most of the application development effort could now occur within a visual design tool where the programmer would piece together an interface from a suite of available libraries. Customization of the components can be accomplished by modifying a corresponding collection of properties sheet. Any “work” to be done within the application is triggered via events that affect the interface components (mouse clicks, keyboard entries and so on). With these first-generation visual development tools, the specification of the actions to be taken on these events is defined using a Basic-like scripting language.

As proven by their broad popularity, these tools went a long way toward solving some of the problems of corporate developers. However, there remained a serious deficit in their capabilities, due to their reliance on the visual design process for creating the application and also their underlying interpreted languages.

As the demands of Windows applications buyers continued to grow, developers were stretching the limits of the existing technologies to create projects such as mission-critical client/server applications. Team development and software quality assurance issues were becoming prominent. Applications were being designed as a series of modules that would need to interact seamlessly, and capable of communicating with and invoking other applications. For example, a user may have a need to insert a graphic in the context of an application. The graphic would be found by accessing a database created by some other application potentially residing over a network on a remote system. All of this needed to be transparent to the application at hand, so that users need not be concerned with where the graphic came from or how it was created.

Specifically, the following issues thus became crucial to professional applications developers:

- Performance
- Rapid Application Development
- Component Reuse
 - Database Scalability

The next logical step in this evolution is technology that combines the significantly enhanced productivity of modern RAD tools with the power and flexible architecture of proven 3GL compiled languages. The remainder of this paper will contrast how Delphi and Visual Basic address these four key criteria for a robust Windows development system.

It is assumed that readers of this paper are familiar with RAD design concepts, but a detailed knowledge of Visual Basic or Delphi is not necessary in order to understand this paper.

Performance

Performance of deployed applications is a key issue in today's highly competitive software market. Particularly for large, distributed client/server applications, any shortfalls in execution speed become far more apparent, due to higher overall system demands.

Delphi is based upon Object Pascal (a significant extension of the popular Borland Pascal 7.0) whereas Visual Basic uses Microsoft Basic as its underlying language. Delphi's performance is significantly better, simply because it generates compiled executable files, while Visual Basic produces semi-interpreted code. That is, Delphi is built around an optimizing native code compiler instead of the slower interpreted p-code used by products such as Visual Basic. This results in Delphi applications executing 10 to 20 times faster than interpreted code. Delphi's intelligent linker also enables segment optimization, thereby reducing executable file size by as much as 30 percent, which enables faster loading and additional performance gains.

Delphi can compile standalone executable files (.EXEs) as well as reusable Dynamic Linked Libraries (DLLs). For the ultimate in execution speed, Delphi also allows professional programmers to go one step further by writing in-line assembler code, for direct control of the microprocessor.

Other areas in which Delphi displays considerable performance gains over Visual Basic is in database connectivity. The database layer of Visual Basic is implemented via ODBC, as opposed to the more efficient Borland Database Engine used in Delphi (and other core Borland development tools). However, Delphi also supports links to data via ODBC drivers. The high-performance native SQL Links supplied with Delphi Client/Server also outperform comparable Visual Basic SQL connectivity options.

Benchmarks

Delphi's superior performance over Visual Basic becomes immediately apparent when running a few simple benchmarks. Consider the following examples, where a database is filled with items of text representing lastname, firstname, phone and street information. The phone number field is filled with consecutive integers, then the database is re-read and filled with a global array of integers from the phone number field. Finally, the global array is sorted to become reverse-ordered using a comparatively slow bubble sort algorithm.

Similar code can be written in both Delphi and Visual Basic, with the stages of the benchmarks summarized in the following code fragments:

VB - Fill

```
Sub btnFill_Click ()
Dim k As Integer
MaxArray = EdArraySize.Text
For k = 1 To MaxArray
    Data1.Recordset.AddNew
    Data1.Recordset("LastName") = "Smith " + Str(k)
    Data1.Recordset("FirstName") = "Joe " + Str(k)
    Data1.Recordset("Phone") = Str(k)
    Data1.Recordset.Update
Next k
Data1.Recordset.MoveLast
End Sub
```

VB - Read

```
Sub btnSearch_Click ()
Dim k As Integer
Dim n As Integer
Dim s As String
Data1.Recordset.MoveFirst
For k = 1 To MaxArray
    s = edPhone.Text
    n = Val(s)
    Call AppendArray(k, n)
    Data1.Recordset.MoveNext
Next k
End Sub
```

VB - Sort

```
Sub btnSort_Click ()
Dim j As Integer
Dim k As Integer
Dim tmp As Integer
For j = 1 To MaxArray - 1
    For k = 1 To MaxArray - j
        If GlobArray(k) < GlobArray(k + 1) Then
            ' Swap GlobArray[k+] with GlobArray[k] ...
            tmp = GlobArray(k + 1)
            GlobArray(k + 1) = GlobArray(k)
            GlobArray(k) = tmp
        End If
    Next k
Next j
End Sub
```

Delphi - Fill

```
procedure TForm1.Button4Click(Sender: TObject);
var
  k,err: integer;
  s: string;
begin
  val(edDBsize.Text,maxArray,err);
  for k:=1 to maxArray do
    with Table1 do
      begin
        str(k,s);
        Append;
        FieldByName('Lastname').AsString := 'NewGuy'+s;
        FieldByName('Firstname').AsString := 'Paul'+s;
        FieldByName('Phone').AsString := s;
        Post;
      end
    end;
end;
```

Delphi - Read

```
procedure TForm1.btnSearchTestClick(Sender: TObject);
var
  s: string;
  n,err,k: integer;
begin
  val(edDBsize.Text,MaxArray,err);
  Table1.First;
  for k:=1 to MaxArray do
    begin
      s := DBedPhone.EditText;
      val(s,n,err);
      AppendArray(k,n);
      Table1.Next;
    end;
  end;
```

Delphi - Sort

```
procedure TForm1.btnSortArrayClick(Sender: TObject);
var
  j,k,tmp: integer;
begin
  for j:=1 to MaxArray-1 do
    for k:=1 to MaxArray-j do
      if GlobArray[k] < GlobArray[k+1] then
        begin
          { Swap GlobArray[k+] with GlobArray[k] ... }
          tmp := GlobArray[k+1];
          GlobArray[k+1] := GlobArray[k];
          GlobArray[k] := tmp;
        end;
    end;
end;
```


Benchmark Results

The following table shows the results for database tables ranging in size from 100 to 4000 records. The test stages *Fill*, *Read* and *Sort* correspond the code sections described on the previous pages.

(All benchmark times are in seconds.)

# Items	Delphi	VB	Delphi	VB	Delphi	x Sort	VB	Delphi	x Total	VB
	<u>Fill</u>	<u>Fill</u>	<u>Read</u>	<u>Read</u>	<u>Sort</u>		<u>Sort</u>	<u>Total</u>		<u>Total</u>
100	2	2	30	1	0	0	0	2	1.5	3
1000	16	70	6	23	1	22	22	23	5	115
2000	33	141	12	46	4	21	84	49	5.5	271
3000	50	227	17	69	8	23.6	189	76	6.4	485
4000	67	297	23	77	15	19.6	294	106	6.3	668

As can be seen from the results, the resulting Delphi-generated code outperformed the Visual Basic routines, especially in code-bound portions such as the Sort stage, by about 20 times faster. Delphi's database access functionality was also shown to be about five times more efficient than that of the Visual Basic code.

Rapid Application Development (RAD)

The other side of the performance issue relates to the speed of application development, which is crucial for programmers intent on ensuring the fastest time to market for their products. The RAD features of an environment are the key to establishing how easy it is for programmers to progress from initial design and prototyping through to final implementation and deployment.

A modern RAD environment provides developers with several elements that significantly speed the development process over the traditional sequential coding approach. These include:

- A visual design environment;
- High-level building block components (often called “controls”);
 - Contextual access to code segments directly, via objects. In other words, homing in on the specific code relating to a particular object.

Under Windows, the structure of an application is frequently molded around its graphical user interface (GUI), with the behavior of the application triggered by various Windows messages or events. The methodology for RAD flows according to the following outline:

1. The developer creates an empty window or *form* to contain the application’s interface components;
2. The developer selects a *component* from a pool of available components, which are generally displayed as an array of icons. Components are then placed and sized on the form;
3. Relevant properties are set or adjusted for each component, according to the application's requirements;
4. Code is written and “attached” to all relevant events for each component;
5. The application is run within the development environment;
6. The developer can then continue to modify the form design or underlying code until the final working application is completed.

Both Visual Basic and Delphi subscribe to this general methodology, making the products appear deceptively similar. However, there are several key enhancements that Delphi adds to this process, including:

- More built-in controls
- Enhanced icon layout, via a fully-customizable, multi-page (tabbed) *component palette*;
- Extensive gallery of extensible project templates and experts;
- Enhanced object placement capabilities;
- Enhanced modification of property lists;
- Two-way, synchronized code window;
- Shared event functions;
 - Integrated graphical debugging and object inspection

Controls

Visual Basic custom controls are referred to as VBXs, and a limited selection is supplied with Visual Basic itself. Additional controls are sold by third-party manufacturers, although these not only cost additional money but also extend the overall learning curve, due to variations in product styles. To utilize controls for a tabbed folder, notebook, database grid or 3D list box, for example, Visual Basic owners must obtain third-party VBXs. Some of the controls supplied with Visual Basic suffer from memory and other limitations, making it necessary to purchase third-party alternatives.

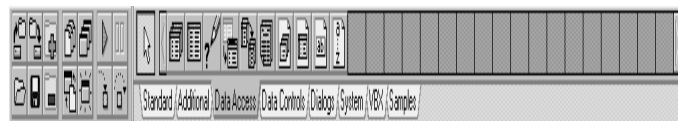
Delphi's Visual Component Library (VCL) is a comprehensive suite of high- performance controls that support all standard Windows functionality, along with additional features such as tabbed folders, notebooks, database grids and 3D list boxes. Delphi also supports third-party VBXs, providing access to a wide range of third-party components.

Control Icon Array

The Visual Basic control display is an array of icons with pictorial representations that are not always intuitive. In other words, developers can be left searching for the *Image Control*, for example, amongst many other icons with a similar look. Developers must then place a control into the form to be certain as to its identity. The Visual Basic control icon array can quickly become unwieldy as additional third-party custom controls are acquired. Since these icons are organized as a configurable rectangle, developers working with a large set of VBX controls are forced to give up valuable screen space or sacrifice accessibility of some controls.



**Visual Basic
VBX icons with
Sheridan (third party)
VBX controls added**



Delphi - Component Palette

Delphi solves these component layout problems with several enhancements. Firstly, Delphi's component palette is organized with several tabbed notebook pages, displaying icons in a single-row, scrollable toolbar format. This keeps the display uncluttered, yet fully accessible. More importantly, Delphi's customization options allow full configuration of the grouping, placement and display of components, so that the environment can be fine-tuned to suit the working style of an individual or development team. To address the problem of obscure or similar-looking icons, Delphi offers "fly-by help" showing the purpose of the control associated with the icon as the cursor is dragged over it.

Templates and Experts

Delphi includes pre-built templates that make it easy to develop standard applications or complex components such as MDI windows, database forms, multi-pages dialog and dual list boxes. The

architecture is fully extensible, allowing developers to easily register their own custom templates and experts into the gallery.

Object Placement

Delphi facilitates visual design with features such as automatic object alignment, sizing and scaling, while Visual Basic supports placement only. Delphi's automatic alignment also speeds up the creation of aesthetic forms.

Property Lists

A subtle yet significant distinction between the two development tools can be seen in the means of accessing property lists. Visual Basic users access a pull-down selection of options for a particular property via an entry bar at the top of the list, so that changing several property items, requires selection of the item, clicking on the entry bar to make the change, then clicking on the next item, and so forth. Delphi provides pull-down lists that can be accessed directly alongside the property value, making for more efficient and intuitive modifications.

Code Window

Delphi's code editing window synchronizes all visual design representations with the underlying source code. In other words, as the application is constructed by dropping objects into a form, the corresponding bug-free code is generated simultaneously. There are no limitations, since the code is always accessible, and developers can instantly switch between the code editor and the visual design tools, allowing them to select the most efficient mode for each part of the project.

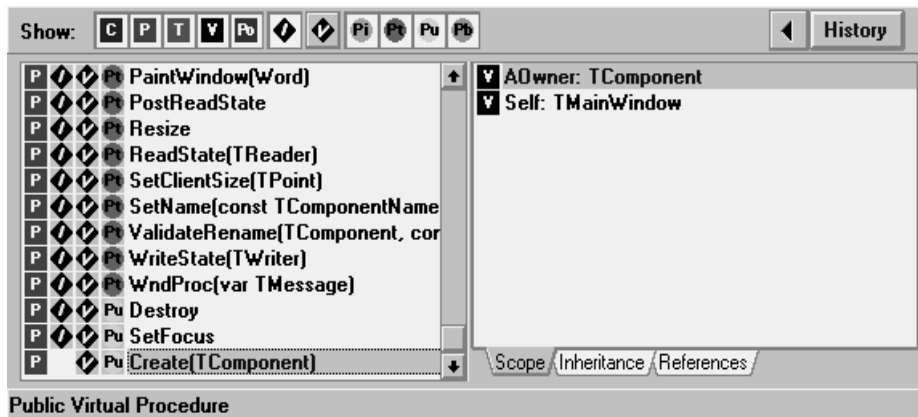
Debugging and Object Inspecting

Visual Basic provides program debugging capabilities such as variable watches and a call stack monitor. However, this functionality is limited in that it cannot break on a specific condition, and the call stack is modal, so it cannot be viewed during the entire debug session.



Visual Basic - Watch Pane

Delphi provides a full-featured debugger with conditional breakpoints and a modeless call stack viewer. The debug window and viewers can be saved from session to session, allowing developers to create a comfortable custom environment. Delphi also includes a powerful object browser similar to that used within Borland C++, which provides a comprehensive display of code objects and classes - including the capability to trace object lineage (inheritance, children) and virtual procedures.



Delphi - Object Browser

Component Reuse

One of the most significant advances in applications development methodology is the concept of creating an application from high-level components. By linking predefined building blocks, developers need only define the “glue” between objects that specifies the unique qualities of an application, with the potential for substantial productivity gains. Although Delphi and Visual Basic both provide various ways to reuse and share components and code, Delphi again delivers a cleaner and simpler solution.

The issue of reuse can be viewed in three areas:

- shared event functions
- reusable functions
 - reusable building blocks (components)

Shared Event Functions

A common problem encountered in Windows programs is how to share a function that is executed upon the occurrence of several Windows events. Although the implementation is similar in both Delphi and Visual Basic, the Delphi solution has some obvious advantages.

In Visual Basic, shared functions must be placed in the local code file or in a global .BAS file if the function is to be shared. The problem with this is that the function is now global to the entire project. In contrast, Delphi allows the function to be placed in the local file or in a DCU (Pascal unit file) which must be explicitly referenced only by the files that use it.

Reusable Functions and Libraries

In Visual Basic, common functions or libraries are accomplished by use of a global .BAS file, which then makes the functions accessible to every file in the project. The disadvantage to this approach is that the shared functions must be global to every file. Alternatively, Visual Basic can take advantage of functions organized in a DLL, but DLLs (like VBXs) must be created by another development tool external to Visual Basic, which requires a different level of expertise and an additional learning curve.

All libraries for Delphi can be created from within the Delphi environment. Pascal code is organized as *units*, and shared functions are accessible through a Pascal unit by simply referencing the “library” unit that contains the desired function. Delphi can also use and create high-performance Windows DLLs. Further, Delphi’s underlying programming language allows developers to reuse and customize functionality within a class via subclassing (see further details of OOP methodology below).

Components

VBXs can be developed for Visual Basic with functionality that is usable across different projects, but a significant disadvantage of VBXs is the complexity involved in creating them. There is a detailed set of restrictions associated with creation of a VBX such that they cannot be created within Visual Basic itself. Instead, the most common method to create a VBX is to use a C/C++ compiler to create a DLL and then put a VBX “wrapper” around it. The advantage of this is the speed of computation gained by using optimized C/C++ compiled code over the Visual Basic’s interpreted technology. The disadvantage is that developers are forced to “switch gears” in order to work with the compiler, and the added complexity can lead to additional troubleshooting and debugging time.

Delphi components are more easily created. Unlike Visual Basic, where VBXs must be built using an external compiler, Delphi components are built within the Delphi development environment itself. This is an important distinction, because professional developers prefer to work with a consistent set of tools. Being able to use Delphi to create reusable components becomes a major productivity enhancement, enabling more rapid development with the added benefits of reusability.

Additionally, since Delphi components are created with Delphi’s optimizing native code compiler and linker, there is improved performance over traditional VBXs. One other considerable advantage of Delphi components is that developers can subclass the functionality of a component to create their own custom versions. If a specific VBX is insufficient for a Visual Basic user, the only alternative is to build (or purchase) an alternative VBX.

Programming Language

An obvious difference between Visual Basic and Delphi is the underlying programming language. The use of Object Pascal within Delphi has several important repercussions:

- Pascal is a more powerful and structured language than Basic.
- Object Pascal is a true object-oriented programming language, providing the benefits of inheritance, encapsulation and polymorphism;
- Pascal is a compiled language, ensuring high-performance executables;
- The organization of files as DCUs provides a cleaner mechanism for creating libraries of reusable code (see *Shared Event Code*);
- Object Pascal utilizes the world’s fastest commercial compiler technology;
 - Object Pascal support in-line assembler code for maximum performance;

One final point of differentiation is that in Visual Basic, all code files must be specifically associated with a form, except for a global .BAS file. In other words, a function must be global to the entire project unless associated with a form. In Delphi, however, code files (and therefore classes and functions) can be disassociated with any form, allowing proper scoping of functions without any loss of functionality.

OOP Design Methodology

The power and flexibility of an object-oriented design methodology is widely accepted as the best way to solve complex, real-world programming problems. Object-oriented design provides both a solid foundation and elegant architecture for an application. Some of the benefits of OOP are:

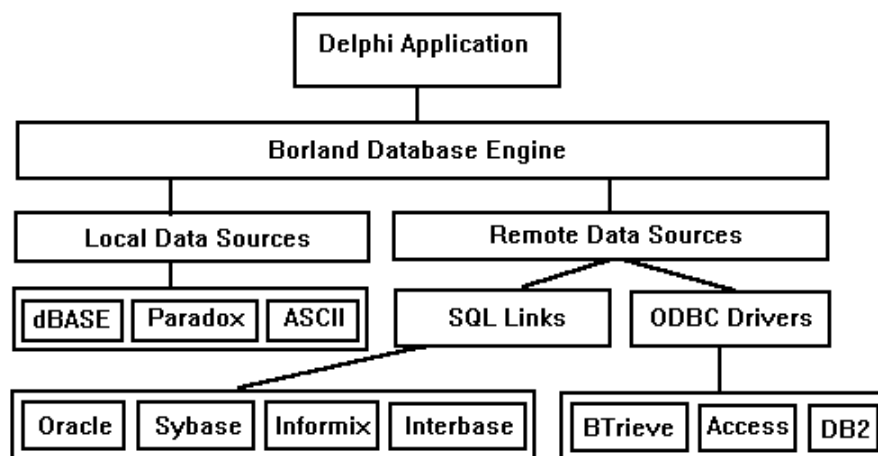
- Shorter development cycles;
- Code that is highly maintainable;
- Code that is easily shared with other modules or other projects;
- Facilitation of team programming and version control;
- By exploiting object inheritance and polymorphism, the coding process can become much simpler and the code itself significantly more coherent;
 - Applications can incorporate several functions that are mostly similar but have certain distinct "personality traits".

Object Pascal is a structured, object-oriented programming language, providing full support for class architectures, inheritance, virtual functions and polymorphism. Visual Basic is not an object-oriented language. Note that although developers need not be familiar with object-oriented concepts to create programs using Delphi, professional programmers will appreciate the benefit of these capabilities.

Database Scalability

A good RAD environment must address the pervasive issue of creating a database application, and Visual Basic and Delphi are no exceptions. In Visual Basic, developers can place a database component onto a form which can then have a property set that allows it to bidirectionally communicate with an ODBC-compatible database. The database component can be used as a crude mechanism to navigate through the database using arrows representing first, next, previous and last records. SQL queries can also be defined in code, to form a query snapshot into the database for viewing or computation. Crystal Reports is shipped with Visual Basic, providing a report generating capability. Setup of the database structures, the associated forms, interaction between them and most of the navigation through the database must all be done explicitly via the visual design tools or within code.

Delphi includes extensive database support including the Borland Database Engine (BDE) for Paradox and dBASE access, and middleware layers that support local and remote SQL data access. The Borland database architecture provide developers with high-performance access to a variety of data sources including ODBC drivers. Delphi includes data access components and data-aware user interface components to provide a comprehensive database solution.

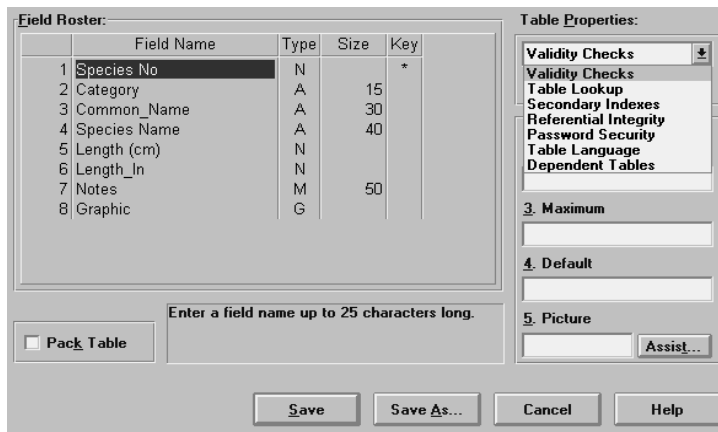


Delphi ships with several controls for data entry and display, including tables and grids. The grid control (TDBGrid Component) can be used to build a spreadsheet-style of application. A unique characteristic of the database grid control is that it can be linked to multiple database sources.



Delphi - Data Base Controls

Delphi also includes wizards and experts that facilitate rapid design and implementation of databases and the corresponding user interface. The DataSet designer facility included with Delphi allows developers to rapidly create table or query data for database components. It is a simple matter to specify which set of fields from the database must be incorporated into the table or query.



Delphi - Data Base Desktop Designer

When designing a database grid, an application often needs an editor to allow in-place modification of field data. Delphi's DBEdit provides a consolidated component to handle this task. Grid-aware, specialized versions of the control are available for labels, lists, combo boxes, images, memo (multi-line editors), check and radio buttons, lookup lists and lookup combo lists.



Visual Basic - Data Manager

Delphi also features built-in support for queries and reports. A query control (TQuery Component) provides the ability to perform SQL queries in order to form the data set corresponding to the filtered elements of a database. If this data was extracted from dBASE or Paradox, you would also have the ability to modify, insert or delete records. By placing this component into a form that also contains the database component, developers can create a filtered, printable report based on some SQL or query into the data set.

Delphi includes the award-winning ReportSmith report writer for PC and SQL databases. ReportSmith provides an intuitive interface for report creation using live data at design time, and it supports queries, crosstabs, templates, calculations and unlimited report sizes.

Moving Up to Delphi

Visual Basic developers who may be considering migrating their applications to Delphi, you may be concerned about the effort required to migrate existing Visual Basic applications in order to continue project development and maintenance within Delphi. Project migration is actually a fairly straightforward process. A conversion utility is available from EarthTrek, Inc (617) 273-0308 that performs most of the translation including project files, form files and code translation. The utility completes all of the possible automatic translation leaving some ambiguous language elements to be identified by a simple syntax check using the Delphi compiler. Many projects can be translated with virtually no effort. Others may require a few hours of post-work to complete.

Conclusion

When examining the various RAD products in the marketplace, both Visual Basic and Delphi stand out as leading edge products. However, Delphi has clearly emerged as a next generation tool with its higher performance, highly facilitated visual design capability, extensive support for reusable components and database readiness. Delphi achieves its goals as a powerful application development system by combining a state-of-the-art visual design environment with the power, flexibility and reusability of a fully object-oriented language, the world's fastest compiler, and leading-edge database technology. Further, through the integration of Object Pascal, Delphi empowers developers with a full-featured programming environment without sacrificing rapid visual development thus allowing construction of sophisticated client-server applications in record time.