Hello again! I bet you are surprised to see an issue again so soon… Well, I was just getting swamped with material and had some free time on my hands (my contract with PG&E ended), so I sat down and figured I would get ahead on UNDU.

I have noticed that sometimes people will re-subscribe to the UNDU subscriber list after they are already on. It highlighted to me a few things I need to say about the way this list works….

First, this is not a listserver that is managing this system. It is a Delphi application that I wrote that looks into my Compuserve mail folders and extracts names and return addresses from people who have sent me an email with the text "SUBSCRIBE UNDU" in the message subject line. It also sends out the notifications as well. Since it is mostly an automated system, it is extracting your return address from the message you send. As a result, the subscriber notification will go out to you on the same email account that you used to send the message to me. Sometimes people will include text saying *"please use this email account instead…".* This is fine if I happen to see it, but often times I dont.

Also, each time I send out the notifications, I invariably will get close to 40 or 50 pieces sent back as "unable to deliver". Usually, this means the users email account is no longer valid. When this occurs, I take that name off the subscriber list to avoid future delivery failures. If you stop getting notifications, you may want to re-subscribe. I can tell the difference between a mail delivery "failure" and a "delay", so I do try over again on those that dont work because of some server problem. And if anyone is interested, there currently is a tad over 1,900 names on the list.

I hope this helps clarify things… if you have further questions, please feel free to contact me!

The randomly chosen winner of this months UNDU Prize is Andrew Jeffries for his article on Getting Control of the Control Panel. His prize is a copy of the book **Delphi-In-Depth** published by McGraw-Hill.

This issue is a "Tools" issue, so we have a bunch of reviews and product announcements!

- Robert

Getting Control of the Control Panel

A Review of the Programmers Guild Animated Tray Icon

A Review of MicroEdge's Visual SlickEdit for Delphi

A Review of IniOut - by Alan Moore

A Review of Addict 2.2 Spell Checker for Delphi

Announcement - SQLExpress Now Available

Delphi Users Groups

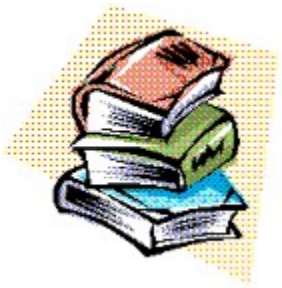
Tips & Tricks

Questions From UNDU Readers

UNDU Subscriber List

Index of Past Issues

Where To Find UNDU



Index of Past Issues

Below is a complete index of all principle articles in past issues of the Unofficial Newsletter of Delphi Users. Provided that you have the prior issues in the same directory as this issue, you can click on any of these hotspots to go directly to that article. To return to the index, you can click on the **Back** button, or you can use the **History** list. Once you jump to one of these issues, you can navigate through the issue as you would normally, but you will need to go to the **History** list to get back to this index. There will be an updated index included in all future issues of UNDU.

Issue #1 - March 15, 1995

- What You Can Do
- Component Design
- Currency Edit Component
- Sample Application
- The Bug Hunter Report
- About The Editor
- SpeedBar And The ComponentPalette
- Resource Name Case Sensitivity
- Lockups While Linking
- Saving Files In The Image Editor
- File Peek Application

Issue #2 - April 1, 1995

- Books On The Way
- Making A Splash Screen
- Linking Lockup Revisited
- Problem With The CurrEdit Component
- Return Value of the ExtractFileExt Function
- When Things Go Wrong
- Zoom Panel Component

Issue #3 - May 1, 1995

- Articles
- Books
- Connecting To Microsoft Access
- Cooking Up Components
- Copying Records in a Table
- CurrEdit Modifications by Bob Osborn
- CurrEdit Modifications by Massimo Ottavini
- CurrEdit Modifications by Thorsten Suhr
- Creating A Floating Palette
- What's Hidden In Delphi's About Box?
- Modifications To CurrEdit

Periodicals
Progress Bar Bug
Publications Available
Real Type Property Bug
TIni File Example
Tips & Tricks
Unit Ordering Bug
When Things Go Wrong

Issue #4 - May 24, 1995

Cooking Up Components
Food For Thought - Custom Cursors
Why Are Delphi EXE's So Big?
Passing An Event
Publications Available
Running From A CD
Starting Off Minimized
StatusBar Component
TDBGrid Bug
Tips & Tricks
When Things Go Wrong

Issue #5 - June 26, 1995

Connecting To A Database
Cooking Up Components
DateEdit Component
Delphi Power Toolkit
Faster String Loading
Font Viewer
Image Editor Bugs
Internet Addresses
Loading A Bitmap
Object Alignment Bug
Second Helping - Custom Cursors
StrToTime Function Bug
The Aquarium
Tips & Tricks
What's New
When Things Go Wrong

Issue #6 - July 25, 1995

A Call For Standards
Borland Visual Solutions Pack - Review
Changing a Minimized Applications Title
Component Create - Review
Counting Components On A Form
Cooking Up Components
Debug Box Component
Dynamic Connections To A DLL
Finding A Component By Name
Something Completely Unrelated - TVHost
Status Bar Component

The Loaded Method
Tips & Tricks
What's In Print

Issue #7 - August 31, 1995

ChartFX Article
Component Cookbook
Compression Shareware Component
Corrected DebugBox Source
Crystal Reports - Review
DBase On The Fly
Debug Box Article
Faster String Loading
Formula One - Review
Gupta SQL Windows
Header Converter
Light Lib Press Release
Limiting Form Size
OLE Amigos!
Product Announcements
Product Reviews
Sending Messages
Study Group Schedule
The Beginners Corner
Tips & Tricks
Wallpaper
What's In Print

Issue #8 - October 10, 1995

Annotating A Help System
Core Concepts In Delphi
Creating DLL's
Delphi Articles Recently Printed
Delphi Informant Special Offers
Delphi World Tour
Getting A List Of All Running Programs
How To Use Code Examples
Keyboard Macros in the IDE
The Beginners Corner
Tips & Tricks
Using Delphi To Perform QuickSorts

Issue #9 - November 9, 1995

Using Integer Fields to Store Multiple Data Elements in Tables
Core Concepts In Delphi
Delphi Internet Sites
Book Review - Developing Windows Apps Using Delphi
Object Constructors
QSort Component
The Component Cookbook
TSlideBar Component
TCurrEdit Component

The Delphi Magazine
Tips & Tricks
Using Sample Applications

Issue #10 - December 12, 1995

A Directory Stack Component
A Little Help With PChars
An Extended FileListBox Component
Application Size & Icon Tip
DBImage Discussion
Drag & Drop from File Manager
Modifying the Resource Gauge in TStatusBar
Playing Wave Files from a Resource
Review of Orpheus and ASync Professional
The Component Cookbook
Tips & Tricks
UNDU Readers Choice Awards
Using Integer Fields to Store Multiple Data Elements in Tables

Issue #11 - January 18th, 1996

Core Concepts With Delphi - Part I
Core Concepts With Delphi - Part II
Dynamic Delegation
Data-Aware DateEdit Component
ExtFileListBox Component
DBExtender Product Announcement
Dynamic Form Creation
Finding Run-Time Errors
Selecting Objects in the Delphi IDE
The Beginners Corner
The Delphi Magazine
Top Ten Tips For Delphi
The Component Cookbook
Tips & Tricks
The UNDU Awards

Issue #12 - Feburary 23rd, 1996

The Beginners Corner
Delphi Projects
Marketing Your Components
An LED Component
A 3D Progress Bar
Common Strings Functions
Checking if your application is running already
AutoRepeat for SpeedButtons
Form and Component Creation Tip
Detecting a CD-ROM Drive
Drawing Metafiles in Delphi
Shazam Review
Product Announcement - Dr. Bob's Delphi Experts
Book Review - Instant Delphi Programming
Tips & Tricks

The Component Cookbook

Issue #13 - May 1st, 1996

Core Concepts - Sorting
Delphi Information Connection
Creating Resource-Only DLL's
Quick Reports
TIFIMG Product Announcement

Issue #14 - June 1st, 1996

A 3-D Component
An Animation Component
A Bug In TGauge
The Component Cookbook
A Look At Cross Tabs
New Book - Delphi In Depth
New Book - The Revolutionary Guide to Delphi 2
Making the Enter Key Work Like the Tab Key
Jumping Straight to Form Level
Making Menu Items Work Like Radio Buttons
Modifying The System Menu
Products & Reviews
The Beginners Corner
The UNDU Awards
Tips & Tricks

Issue #15 - August 1st, 1996

UNDU - A Work In Progress…
UNDU Prizes!
The UNDU Subscriber List
Core Concepts With Delphi - Parameter Passing
Delphi Programmers Book Shelf
Component Cookbook
Tips & Tricks
How to 'Catch'Keys
Working with String Grids
Coloring Columns in a Grid
Solving a DLL problem
Reducing Memory Requirements
Creating an AutoDialer component

Issue #16 - September 1st, 1996

Menu Buttons
Core Concepts With Delphi - Enumerated Types
Extending The INI Component
Limiting Multiple Instances Of a Program in Delphi 2.0
How to Draw a Rubber-Banding Line
Marching Ants!
How to Restrict the Mouse Cursor
How to make a Color ComboBox
A Better Way to Create Menu Items
Splash Screen

Splash Screen with a Time Delay

Issue #17 - October 1st, 1996

Does Windows 95 give you a Square Deal?
The Great StringList
Manipulating Regions with Delphi
Tips & Tricks
When Delphi's smart-linker doesn't seem so smart
Cut, Copy, & Paste
A Quick Way of Setting the Tab Order
Background Bitmaps on Forms
Non-Rectangular Windows

Issue #18 - November 1st, 1996

Object Express by OOPSoft Inc
Tips & Tricks
The Component Cookbook
IniOut Component Property Manager
New Book - Delphi Component Design by Danny Thorpe
Storing Fonts in INI Files
Sorting Columns in a DBGrid
What's Your Version Number
Drawing MetaFiles
Adding Undo to your Edit Menu
How To Put Anything In Your Delphi EXE
Delphi Newsgroups
A Simple Clipboard Viewer Component

Issue #19 - January 1st, 1997

Speed Daemon Review
A Look at MagiKit
Humor - Are You Computer Illiterate?
Tips & Tricks
The Component Cookbook
Using the SHFileOperation to Copy/Move/Delete/Rename Files
How to create a Polygon Splash screen
Is Someone else running?
Lock Violation
Printing Directly to a printer
Refreshing MDI Menus
Extending the Background Bitmap Technique
Paradox File Size Limits
Safer use of Enumerated Types
Simplifying Code management with Include
A Look at the TreeView Control
Text, Aligned in a Grid
TPageControl Flambé
Big Bitmaps
Masks ala Transparency

Issue #20 - March 1st, 1997

Learning How To Drive - Disk Information in Delphi

Delphi Books & Periodicals
Questions (and Answers) From Readers
Tips & Tricks
The Component Cookbook
Is Someone Else Running - Revisited!
InputQueryEx
Multi-colored text in a string grid
Converting Pascal Source to HTML
Processing large database tables
SHFileOperation Revisited
How to Make Your EXE's Lighter!
Form Aspect Ratio
Previous Instances Revisited
Printing Raw data to the Printer
Tip Of The Day Component
TFieldPanel

Issue #21 - May 1st, 1997

Video Capture in Delphi
Review - Delphi Component Design
Product Announcement - Addict for Delphi
Questions (and Answers) From Readers
Tips & Tricks
The Component Cookbook
Low Level Windows Stuff
Listing Procedures
Hiding Apps from the Task Bar
Excel OLE Tips
Playing Sounds Asynchronously
Bitmaps on StringGrids
How To Find Up-to-date Delphi 2 Books
Margin Marker in Delphi 1 & 2
Lock Violations
Object Creation Tip
How to Compress a Bitmap
TEndSession

Return to Front Page

Where To Find UNDU

When each issue of UNDU is complete, I put them in the following locations:

1. UNDUs official web site at `http://www.informant.com/undu/index.htm`. This site houses all the issues in both HTML and Windows HLP format. Click on the large icons for the HTML versions and the small red book icons for downloadable Windows HLP files.
2. *Borlands* Delphi forum on CompuServe (**GO DELPHI**) in the "Delphi IDE" file section. This forum will only hold the issues in Windows HLP format.

Tips & Tricks

Last issue there was an article on how to insert Bitmaps in String Grids. I should be flogged repeatedly with a wet noodle for not catching the problems with this one. It was unnecessarily complex and many of you let me know it. Lets try it one more time with Bitmaps in String Grids.
In a couple of recent issues I had a discussion on preventing multiple instances of an application from running at once. There had been one pesky little problem however, but I finally came up with a solution… Check it out in A Cure For Previous Instance Problems.
Arlan Mock brings us two handy tricks this month with his articles on Word Search Tip and Unbridled Acceleration

Have you ever noticed in some editors (such as the Delphi IDE or MS Word) that it shows the line and column number at the bottom of the screen as you type. Well, Rolando Casumpang came up with an interesting technique to accomplish something similar in Showing The Caret Position. It isn't a perfect solution but maybe it can serve as a springboard for further experimenting.

Have you been using INI files simply because using the Registry sounds too complex. Well you don't have an excuse anymore. Wim Vandersmissen shows us step by step how it is done in Using the Windows 95 Registry.

Also a while back, we had a few discussions on wallpapering forms. One thing eluded us though… How do you do it with an MDI application? Well know you can courtesy of Mark Pritchard in Wallpapering MDI Forms.

Back in issue #20, Gene Fowler presented an interesting technique for adding a history list to the InputQuery function in Delphi. Well, he revisits the example to make a Delphi 3.0 compatible version in his article this month on InputQueryEX.

Have you ever run into problems with really large projects in Delphi? Check out the discussion on Exceeding the Data Segment.

And lastly… Have you just recently converted to Delphi 3.0? Still want to keep Delphi 2.0 on your system? Well you can make the process a bit smoother with the tip from Kevin Gallagher in his article on Delphi 2.0 and Delphi 3.0 on the Same Computer.

Return to Front Page

UNDU Subscriber List

The subscriber list is a method by which I can notify the readers when a new issue is out. I will maintain a list of readers email addresses and when a new issue is released, I will fire off a batch mailing to notify everyone that it is available.

This is what you need to do to get on the subscriber list… Simply send me an email to my CompuServe address (`RobertV@compuserve.com`) and put the words **SUBSCRIBE UNDU** anywhere in the subject line or in the main body of the message. If you no longer wish to be notified of future issues (i.e. you are on the list and want off…) just send an email with the words **UNSUBSCRIBE UNDU**.

Thats all there is to it!

Return to Front Page

Questions (And Answers) From UNDU Readers

I often get a wide variety of emailed questions from readers of UNDU. Some of them have been quite interesting and the solutions are equally interesting. Anyway, I figured "Why not let everyone help on the solution?"

Each month I will present a few questions here that readers have submitted to me and open them up to all the readers of UNDU. If you know the answer to a question, feel free to send it in to **RobertV@compuserve.com**. I will chose the best solution to the question and post it in the following issue. This way, everyone gets to see the answer!

The solutions can be anything including even shareware components that might solve a particular problem.

This Months Questions Are:

Neil Howard at i-am-de-walrus@geocities.com asks:

"How do I format disks in Delphi without shelling to DOS?"

Return to Front Page

A Review of IniOut

by Alan C. Moore, Ph.D.

Users expect a lot from the applications we write for them. They had better be fast, feature-rich, and free of bugs. Today users have also come to expect some level of flexibility and control, particularly in terms of the configuration of the user interface. In other words, they want to be able to adjust certain aspects of an application's appearance and they might want certain field values to be "remembered" from one run to the next.

Delphi's component properties and Windows configuration files (*.INI for 16-bit versions, the registry for 32-bit versions) help a lot. Still, there's a certain amount of grunt work involved in monitoring certain component properties and writing them to and reading them back from configuration files. Wouldn't it be great if there were an easy way to manage this so that we would simply tell the application which component properties to monitor and then with just a few lines of code? Now we can! Enter Robert Vivrette's IniOut Component.

This clever and useful component belongs to the class of helper components that Ray Lischner calls metacomponents in "Secrets of Delphi 2", i.e. components which manage other components. In this case, having set up your form with all of its various components you simply add a TIniOut component to it. Then if you double-click on it (or click on it in the Object Inspector) you'll get its specialized dialog box (see Figure A) which allows you to manage all of the component properties you need to save between application runs.

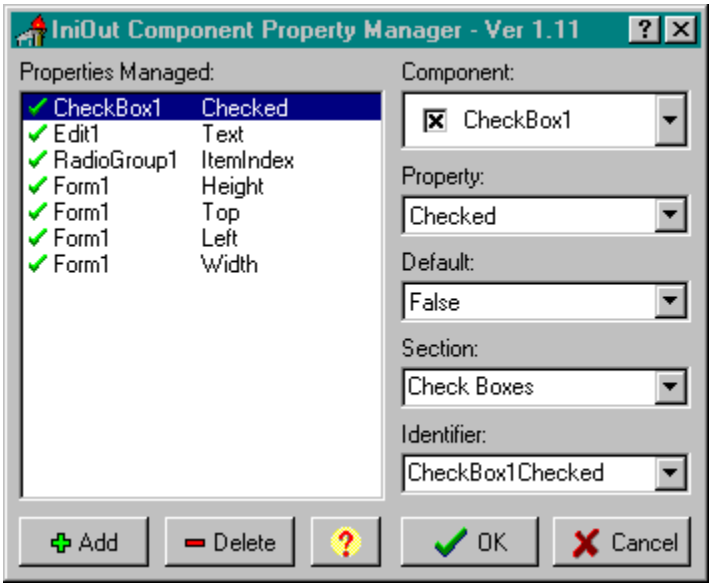


Figure A

In the test project I wrote I used both data settings (TEdit text and TRadioButton selections) and several aspects of the visual design (the main form's Top, Left, Height, and Width properties) that I wanted to restore the next time I fired up the application. It worked-- TIniOut did its job flawlessly! Figure B shows the application with new data settings and in a different location (not shown) from the original settings. Further, an examination of the Windows Registry shows that all of the settings are properly registered there (see Figure C.) And TIniOut automatically set up all of the subkeys for me. Very cool.

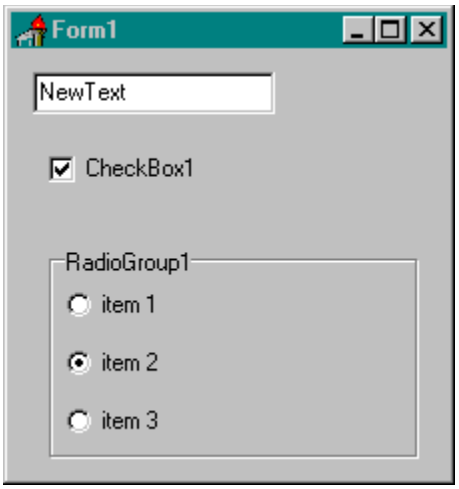


Figure B

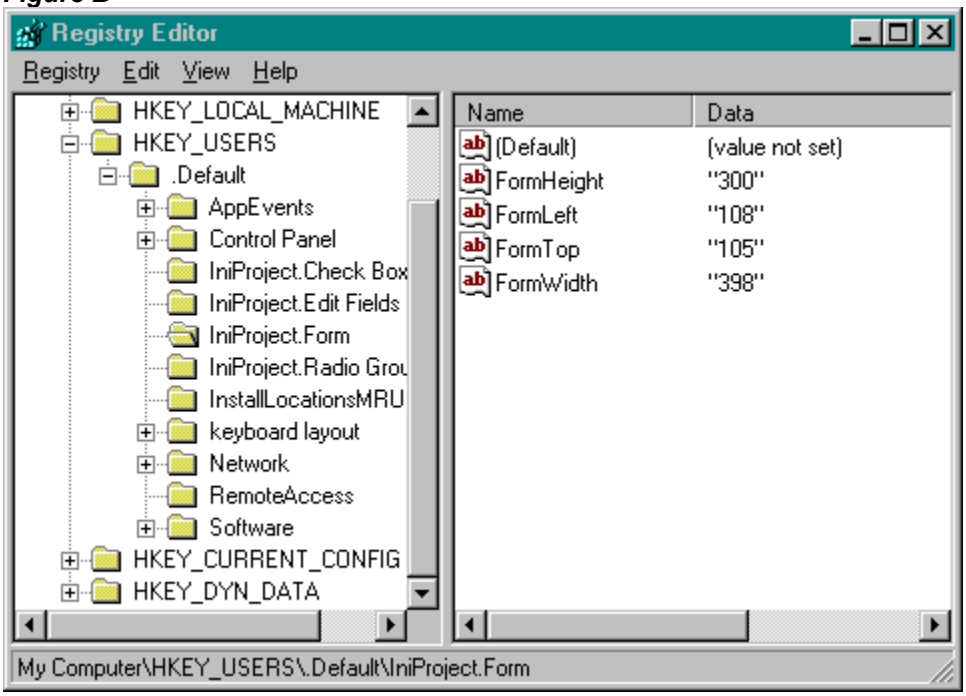


Figure C

With all of its powerful features for maintaining configuration files for our applications, I would recommend this component highly to any Delphi developer. Further, how can you lose with shareware? You can't. So, download TIniOut, convince yourself that I am not exaggerating about its usefulness, and then go ahead and get the registered version, which costs $20 (an absolute bargain, in my opinion.) You can register 32bit version on Compuserve at SWREG #14633, and the 16bit version at SWREG #14724. The 16-bit version can be found at http://www.informant.com/undu/iniouts1.zip and the 32-bit version can be found at http://www.informant.com/undu/iniouts2.zip.

There is a special registration option available with includes the source code for $99. In my view this price is much too high, despite the considerable merits of the source code. Here's what I've suggested to Mr. Vivrette: sell the component only with source code and raise the price to $39 (still a good deal, particularly with the code included.) There's a lot of reluctance on the part of many Delphi developers (including me) to purchase any component without its source code. Further, I think a lot of adventurous programmers could benefit from studying this code. Maybe you've read what Ray Lischner has to say about RTTI (run time type information), specialized property editors, and metacomponents in "Secrets of Delphi 2" but are still wondering how to use this knowledge in your own custom components. TIniOut provides a perfect model.

As of this writing, the Delphi 3 version of IniOut is available but has not been released as shareware. If registered users of IniOut had the source code, they would already have been able to "package" the component for use in this latest Delphi version. Also, making the code available to the Delphi community (at a reasonable rate) might encourage others to develop similar useful components. I understand that there will be some changes in the pricing structure in the near future -- stay tuned for developments.

For any Delphi developer who needs to save configuration information, I recommend this component highly. It comes with excellent documentation in its help file, and its creator is committed to improving it based upon users' input. Examples? In addition to Delphi 3 support, here are some of the enhancements Mr Vivrette has planned for future versions:

- The ability to add managed items (nonpublished component properties, global and local variables) at runtime.
- Support for String lists
- An Editor for default fonts, sets, and colors
- The INI file can reside in any directory and not be required to be in System directory
- The ability to set some properties as readonly (i.e. they would not write their values when SaveToIni or SaveToRegistry is called).
- Boolean values would be saved as 1/0 instead of True/False for backward compatibility

So my concluding advice is simply to download this component, try it, then get the registered version; you'll love what it does for you and your users.

About Alan Moore

Alan C. Moore is a Professor of Music at Kentucky State University, specializing in music theory and composition. His experiences in programming go back to working with Fortran as a graduate student at Yale in the mid 1960's and more recently developing educational software with the Borland languages. He writes regularly for a number of technical journals including Delphi Developers Journal and Delphi Informant. With 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.

Return to Front Page

A Cure for Previous Instance Problems

by Robert Vivrette - RobertV@compuserve.com

In both issues #16 and #20 I presented a discussion on how to restrict multiple instances of an application running at once. There was one problem, however, that was common to both discussions, namely that when you used the technique in the Delphi IDE, it counted the design-time form as one of the instances. As a result, the program would refuse to run if run from within the IDE.

After thinking about it a bit, I came up with a simple solution. In your design-time copy of the form, delete any text in its caption property. Then, in the form's FormCreate method, assign whatever caption you want (i.e. Caption := 'Form1'). Because the technique limits previous instances by looking at the window titles, the design-time form (that doesn't have the appropriate title) will not be viewed as another instance.

Problem solved!

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #22 - June 1997

A Review of MicroEdge's Visual SlickEdit for Delphi

by Robert Vivrette - RobertV@compuserve.com

Borland is in the business of making hot programming languages. They excel at fast code, fast compilers, and fast application development. However, quite frankly, they are not in the business of making full-featured programming editors. The IDE code editors in Delphi 1.0 and 2.0 are fairly basic. They perform the necessary functions that a programmer needs, but not many frills. Delphi 3.0 has listened to the screaming of the Delphi community and has added quite a few extensions to the way the code editor behaves, all under the name "Code Insight". Code Insight includes 4 key features, namely Code Completion, Code Parameters, Tooltip Expression Evaluation, and Code Templates. All of these go a long way towards making the code editor a more functional and efficient environment to work in.

Enter MicroEdge's Visual SlickEdit (or VSE)…

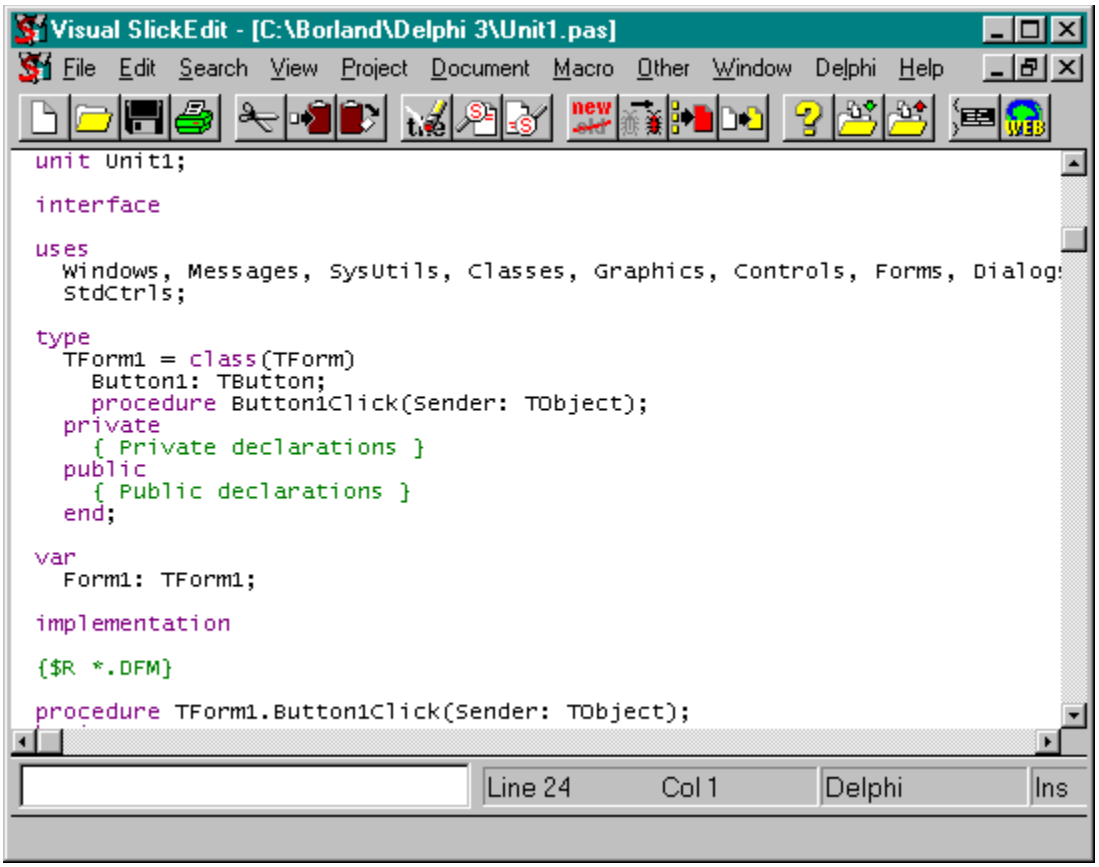
Simply stated, VSE is a programmers editor. It has been around for a number of years in various forms and has quite a faithful following. Its roots originate in the C/C++ developer community and you can see this throughout the application. It supports a dizzying array of languages and operating systems. Internally, it supports C++, C, Delphi, Java, HTML, Pascal, Assembly, and Slick-C (MicroEdge's macro language). It can also install support for AWK, Perl, Modula-2, dBASE, COBOL, FORTRAN, REXX, or Ada. MicroEdge has versions of VSE that run on Win95, Win31, WinNT, OS/2, SCO, Unixware, Sun OS, Solaris, HP-UX, AIX, Linux, UNIX, and a few more I haven't even heard of before.

With the introduction of VSE 2.0, MicroEdge has added specific support for Delphi. Initially, I was having some difficulty getting them to work together, but a recent update from MicroEdge added support for Delphi 3.0, and resolved all my previous problems. This update is available on their Web site and automatically updates an existing VSE 2.0 installation.

So… down to brass tacks. How does VSE work with Delphi? Well, the integration is a kind of "Love/Hate" relationship so it is fairly difficult to really give a definite opinion. Let's go through a bit of how this integration works, and you will see what I mean. For the purpose of this review, I am going to be primarily discussing the integration of VSE with Delphi, rather than a review of all of VSE's extensive capabilities. I will be using Delphi 3.0 in this discussion.

For the two programs to work together, they need to establish a "connection". This is a fairly straightforward and simple process. Basically, if you are running Delphi, you can pick "Visual SlickEdit" from the tools menu. That will launch VSE and establish the connection. You can also start in VSE and choose "Connect to Delphi 3" from its Delphi menu. That will launch Delphi and establish the connection as well.

Once the connection is established, VSE synchronizes its behavior with what Delphi is doing. For example, if you start a new project from Delphi, VSE will automatically show the same information you see in Delphi's Edit window. If you change text in either program, the change is automatically reflected in the other. MicroEdge is proud of the fact that it can accomplish this without needing to save the files, and many programmers will appreciate the extra work they went to to achieve this as well. I tried very hard to make changes to one side or the other and to get them out of sync, but VSE kept up with every change I made. When you go to Delphi and add components onto a form, VSE shows the textual changes to the form's unit as you would expect. It even keeps the cursor in the same spot so you never have to find where you are when switching between the two.



This, then is the basic capability that VSE adds to your Delphi programming. An alternative editor that keeps up with Delphi's editor. It has other interesting features such as adding commonly used Delphi functionality inside of VSE's Delphi menu. For example, you can start a new Delphi application inside of VSE, or Compile/Build a project as well. There also connections to Delphi's PVCS Version Control System allowing you to check-in and check-out code modules.
Now as for VSE as an editor, I must say it is pretty impressive. If I had to spend all day typing code, I could see where VSE would save me a considerable amount of time. Here are some of the features that I particularly liked:

Word Completion - VSE has the ability to automatically complete words that you have previously typed. For example, if you are creating several procedures and you start typing the word 'procedure' again, you can stop after the first character or two and hit CTRL-SHIFT-COMMA and it will complete the word for you. VSE looks backwards through what you have typed and picks the word that best matches what you just partially typed and completes it for you. Saves a lot of time.

Another feature I particularly liked was VSE's ability to selectively hide and show lines. For example, you can hide all comments and VSE collapses the text, excluding comments:

```
function ShowAMessage;
begin
  ShowMessage('Hi There');
end;
```

By clicking on the icon to the left, the comments come back:

```
function ShowAMessage;
begin
  // This is a single-line comment
  { This is a multi-line
    comment that I have entered}
  ShowMessage('Hi There');
end;
```

VSE also has the ability to collapse the code section of a block, so you can see for example, just the function and procedure headers in a unit without looking at all of the code associated with each unit. I really loved these two features.

As you can see from the partial screen shots above, VSE supports keyword colorizing, which is user definable. It also has an interesting feature called SmartPaste. When you cut or copy a block of text and paste it somewhere else, VSE automatically adjusts its indenting to match the area where it has been inserted. Another great time-saver.

VSE also has a very impressive differencing system. This allows you to take 2 source code files and compare them side by side. It does a very good job of illustrating what has changed in each file, unlike many of the other systems I have seen. It also allows you to easily copy text back and forth between the two files.

The help integration of VSE is also a key selling point for me. When VSE is installed it hunts down programming-related help files on your system and ties them together within its environment. Whenever you hit CTRL-F1 you are shown the various help files that contain that topic, so you can decide which to use.

No One Is Perfect...

There is no question in my mind that VSE is one of the hottest programming editors on the market. It is slick (no pun intended) and **highly** configurable.

However, by its very nature, it has been designed to work with virtually any language, development environment, and operating system. This tends to make its Delphi integration look like only a minor capability in the big scheme of things. There are still plenty of areas that I would like to see more tightly integrated. For example, when you choose the compile command in VSE, it switches over to Delphi and compiles your code. If there are errors, it locates them in Delphi's editor making it necessary for you to manually switch task-switch back to VSE if you want to make the change there. In VSE, I would like to have compiler errors listed and affected lines of code highlighted, but I couldn't accomplish this. MicroEdge indicates this capability is available if you configure the language compiler to output a list of error messages, however if Delphi can do this, VSE doesn't make it easy to get there. That whole process was a bit daunting so I moved on.

There were also some unfortunate conflicts in hotkeys that I discovered while using VSE. For example, in Delphi I am used to hitting F9 to compile and run the application. Thinking that VSE had tied the F9 key to the same feature in Delphi, I tried it out. It turns out though, that the F9 key is the Undo key in VSE. VSE's keyboard mapping is completely configurable, so I am sure this could be corrected. It would have been nice to have it done beforehand however.

One last complaint I have is an odd effect in VSE when you are in Delphi and compile the application. It seems (to me at least) like VSE closes and reopens the editing file under certain circumstances. Initially this made the edit window in VSE close and then reopen in a new location for no apparent reason. I eventually got it to keep the window maximized, but I still from time to time would see it flash indicating it had closed and reopened it again. The integration would "feel" much smoother if this odd visual effect was corrected.

In Conclusion

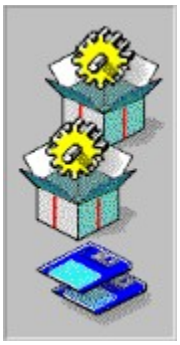
Don't get me wrong. I love VSE, and I may make it my standard editor for Delphi programming. However, tighter integration with the Delphi IDE would really help it out quite a bit. If I had a wish, I would like to see VSE **completely** take the place of Delphi's code editing window. However, because MicroEdge is committed to making VSE as widely compatible as possible, I don't see them spending the time and effort doing this. If they did, I would be first in line to get a copy though.

For a first time user, VSE can be a bit daunting, and the learning curve might be a bit steep. The overwhelming number of configuration options shows how you can really make it do anything you want. But at the same time, this highly configurable nature makes it a little more difficult to really get a grasp on the best way to set it up.

VSE is a very fast and effective code editor for programmers. For those of you who have used VSE in other languages and/or operating systems, the inclusion of support for Delphi is a great selling point for this latest version. No doubt the Delphi support will get even better with future releases.

You can learn more about MicroEdge and Visual SlickEdit at their web site http://www.slickedit.com.

Return to Front Page

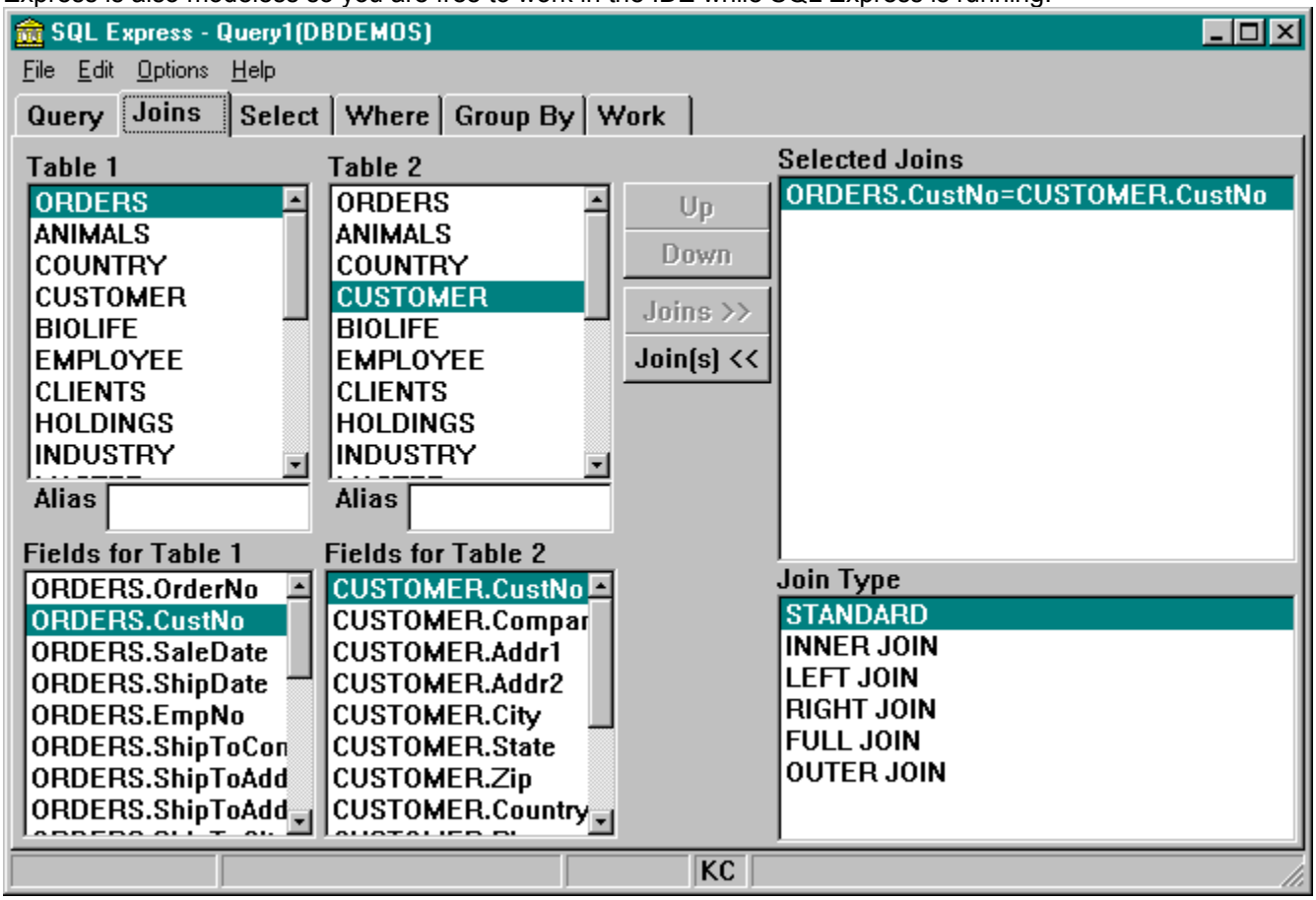


SQLExpress Announcement

from OOPSoft Inc.

Editors Note: Look for a full review of SQLExpress in the next issue of UNDU!

SQL Express is a fully Integrated Query Building tool and Interactive SQL Editor for Delphi and C++ Builder designed around the Borland Database Engine. With SQL Express, the developer can interactively build, test and work with live data. The Interactive SQL Editor allows the developer to run scripts or queries on the database platform they are working. SQL Express promotes minimal typing and in many instances only parameter names, data values and alias names require keyboard action. SQL Express is also modeless so you are free to work in the IDE while SQL Express is running.



SQL Express's advantages include:

Its Live Data! - Instant results from within the property editor. See your data at query design time.

Minimal Typing - Typing is only required for literal values, parameters and alias names. Results are a

mouse click away.

Modeless - SQL Express is modeless. You can switch between SQL Express and the IDE.

Switch databases - You can easily change databases within SQL Express without closing the property editor.

Parameters - Add/Edit your parameters in the same window you design your SQL statements. No need to work with the Params property.

DataType - You know longer have to be bothered with setting the DataType for the Params property in TQuery.

View Image/Memo Field data - View Images and Memo field data live in SQL Express. You can even edit this data at design time!

Alias support - Assign alias names to tables and the change is instantly propagated throughout the query.

Object Creation - SQL Express will generate new TDataSource and TDataGrid objects linked to the current query or a new TQuery object based upon the current TQuery object.

Sub-Select Support in the "IN" clause - SQL Express allows you to create select statements for the "IN" clause.

SQL Elapsed time - Instantly displays the time a query took to execute.

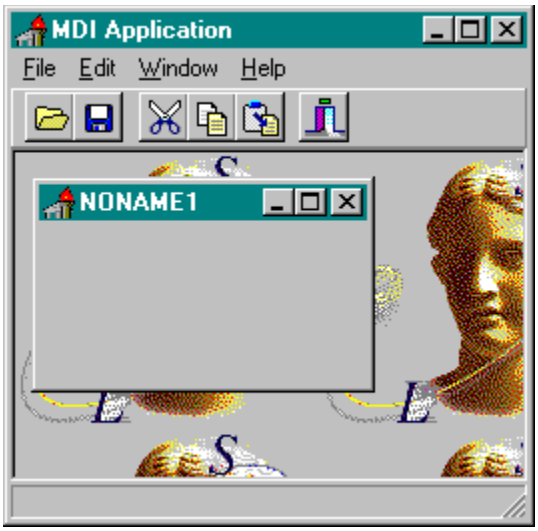
ISQL Editor - Complete ISQL Editor with Boiler Plate Template Support!

Create your own templates - Add your own Boiler Plate Templates for others in your group to use!

ALL AT DESIGN TIME !

For more information on OOPSoft or SQLExpress, you can visit their web site at http://www.oopsoft.com or contact them via email at oopsoft@airmail.net.

Return to Front Page



Wallpapering MDI Forms

by Mark Pritchard - pritchma@netlink.com.au

I notice that UNDU had a wallpaper discussion going for a little while. Well, I wrapped this up into a component, which also saves and loads the forms position to and from an INI file. It works under Delphi 1 and 2, and most likely 3, which I haven't got my hands on yet.

The component is called TIDSFormAssist and serves a dual purpose. First, it saves the forms location and size in an INI file, and more importantly (for this discussion) provides wallpapering, particularly on an MDI form. I hadn't seen anyone else accomplish this technique, but this component does, and is very easy to use.

The properties of interest are Options, Wallpaper and WallpaperMode. For Options, set faoLayout to True to save/restore the position and size of the form. Set faoWallpaper to True to wallpaper the surface of the form. The Wallpaper property Holds the bitmap used to paint the surface. WallpaperMode can be set to wmTile to tile, or wmCentre to centre the bitmap.

I should point out that this component was one of those "you want wallpaper on an MDI form? OK...how long do I have? This afternoon? Cripes!" It works well, but may not be the most elegant solution ;-)

The implementation of this is a little more difficult, since the component also handles the wallpapering of MDI parent forms. It accomplishes this by installing a hook into the form's message processing loop, and watching for the WM_ERASEBKGND message. This seems to be the most reliable and efficient method of painting MDI parent forms, and has the added benefit of working for fsNormal and fsMDIChild forms!

There is a bit of additional work involved in translating the window handle to a TIDSFormAssist instance, but you should be able to sort it out from the source code. If you can't, just close your eyes and forget about what's going on under the hood!

Source Code for Wallpaper Project

Return to Tips & Tricks

Return to Front Page

Source Code for Wallpaper Project (FormAsst.pas)

```
{ TIDSFormAssist - Assists you with common form management tasks.
  PROPERTIES:
    Options         - determines which things you wish to handle automatically.
    faoFormLayout   - Restores and saves Left, Top, Width and Height to the INI file
    INIFile         - specifies the INI file to which the component will restore and
                      save settings if not specified, it is set to Application.EXEName
                      with a .INI extension }
unit FormAsst;

interface

uses
  WinTypes, WinProcs, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
   { Forward class declaration }
   TIDSFormAssist = class;

   { Enumerated types for Options }
   TFormAssistOption  = (faoLayout,faoWallpaper);
   TFormAssistOptions = set of TFormAssistOption;

   { Enumerated type for wallpaper tile mode }
   TWallpaperMode = (wmTile,wmCentre);

   { Type for OnWallpaperPaint event }
   TOnWallpaperPaintEvent = procedure(Sender : TIDSFormAssist) of object;

   TIDSFormAssist = class(TComponent)
   private
       { Internal field storage }
       FOptions            : TFormAssistOptions;
       FOnWallpaperPaint   : TOnWallpaperPaintEvent;
       FWallpaper          : TBitmap;
       FWallpaperMode      : TWallpaperMode;

       { Internal variables }
       { Used to stop recursion }
       blRecurs : Boolean;
       { Whether we should be painting }
       blPaint : Boolean;
       { This component's owner form }
       frmOwner : TForm;
       { The handle of the owner form }
       hwndOwner : HWND;
       { Pointer to previous window handler }
       pPrevWndProc : TFarProc;
       { INI file for layout storage }
       sINIFile : String;

       { Internal routines }
       { Load the form's layout from the INI file }
       procedure LayoutRestore;
       { Save the form's layout to an INI file }
       procedure LayoutSave;
       { Handles a change to the wallpaper }
       procedure WallpaperChange(Sender : TObject);
       { Creates new wallpaper object }
       function  WallpaperCreate : TBitmap;
       { Initialises wallpaper handling }
```

```
       procedure WallpaperInit;
       { Tiles the wallpaper }
       procedure WallpaperPaintTile(dcForm : HDC; rectWnd : TRect; iBmpW,iBmpH :
Integer);
       { Centres the wallpaper }
       procedure WallpaperPaintCentre(dcForm : HDC; rectWnd : TRect; iBmpW,iBmpH :
Integer);
       { Field modification routines }
       procedure SetWallpaper(bmpNew : TBitmap);
       procedure SetWallpaperMode(wmNew : TWallpaperMode);
   protected
       destructor Destroy; override;
       procedure Loaded; override;
   public
       constructor Create(cmpOwner : TComponent); override;
       { Handles the repainting }
       procedure WallpaperPaint(hwndPaint : HWND);
   published
       property OnWallpaperPaint : TOnWallpaperPaintEvent read FOnWallpaperPaint write
FOnWallpaperPaint;
       { Determines which form assistance options to use }
       property Options : TFormAssistOptions read FOptions write FOptions;
       { Wallpaper holds the bitmap which stores the background bitmap }
       property Wallpaper : TBitmap read FWallpaper write SetWallpaper;
       { WallpaperMode handles whether the wallpaper is tiled or centred }
       property WallpaperMode : TWallpaperMode read FWallpaperMode write
SetWallpaperMode;
   end;

   TIDSFormAssistList = class(TComponent)
   private
       lstObjects : TList;
   public
       { Adds a new TIDSFormAssist to the global control list }
       procedure Add(faNew : TIDSFormAssist);
       constructor Create(cmpOwner : TComponent); override;
       destructor Destroy; override;
       { Returns the TIDSFormAssist managing the HWND or Nil if none match }
       function GetFormAssistFromHWND(hwndSrch : HWND) : TIDSFormAssist;
       { Removes a TIDSFormAssist from the global control list }
       procedure Remove(faOld : TIDSFormAssist);
   end;

procedure Register;

implementation

uses INIFiles;

var
   gfalControl : TIDSFormAssistList;

procedure Register;
begin
   RegisterComponents('IDS', [TIDSFormAssist]);
end;

{ Event handler for window moves and sizes - used for background painting }
{$IFDEF VER80}
function FormAssistWndProc(hwndDest : HWND; uiMsg : Word; wpIn : SmallInt; lpIn :
LongInt) : LongInt; export;
{$ELSE}
function FormAssistWndProc(hwndDest : HWND; uiMsg : UINT; wpIn : WPARAM; lpIn :
```

```
LPARAM) : LRESULT; stdcall;
{$ENDIF}
var
   faInst : TIDSFormAssist;
begin
   { Retrieve the form assist managing this hwnd }
   faInst := gfalControl.GetFormAssistFromHWND(hwndDest);
   { Check if we are erasing background, the form assist is valid and has a valid
wallpaper bitmap }
   if (uiMsg = WM_ERASEBKGND) and (Assigned(faInst)) and
      (not faInst.Wallpaper.Empty) then
     begin
         { Paint the window }
         faInst.WallpaperPaint(faInst.hwndOwner);
         { Indicate we have handled the repaint }
         Result := 1;
     end
   else
     { Pass the message on }
     Result := CallWindowProc(faInst.pPrevWndProc,hwndDest,uiMsg,wpIn,lpIn);
end;

constructor TIDSFormAssist.Create;
begin
   inherited Create(cmpOwner);
   { Initialise }
   Options             := [faoLayout];
   pPrevWndProc        := Nil;
   FWallpaper          := WallpaperCreate;
   WallpaperMode       := wmTile;
   { Retreive the INI file }
   sINIFile := ChangeFileExt(Application.EXEName,'.INI');
   { Add ourself to the global control list }
   gfalControl.Add(Self);
end;

destructor TIDSFormAssist.Destroy;
begin
   { Ensure we are in not in design-mode, and our owner is a TForm }
   if (not (csDesigning in ComponentState)) and (Assigned(frmOwner)) then
     begin
       { Save the form layout }
       if faoLayout in Options then LayoutSave;
       { Unhook our message handler }
       if Assigned(pPrevWndProc) then
           SetWindowLong(hwndOwner,GWL_WNDPROC,LongInt(pPrevWndProc));
     end;
   { Destroy allocated objects }
   FWallpaper.Free;
   { Remove ourself from the global control list }
   gfalControl.Remove(Self);
   inherited Destroy;
end;

procedure TIDSFormAssist.LayoutRestore;
const
   cNOTFOUND = -1;
var
   iniRead : TINIFile;
begin
   { Check if position is modifiable }
   if frmOwner.Position = poScreenCenter then
     raise Exception.Create('Layout cannot be restored when Position =
```

```
psScreenCenter');
   iniRead := TINIFile.Create(sINIFile);
   try with frmOwner do
     begin
       { Attempt to read the Left property }
       if iniRead.ReadInteger(ClassName,'Left',cNOTFOUND) = cNOTFOUND then
         begin
           { Layout hasn't been stored - programmatically position it }
           if FormStyle = fsMDIChild then
             begin
               Left := 0;
               Top := 0;
             end
           else
             begin
               Left := (Screen.Width - Width) div 2;
               Top  := (Screen.Height - Height) div 2;
             end;
         end
       else
         begin
           { Read layout from INI file }
           Left   := iniRead.ReadInteger(ClassName,'Left',Left);
           Top    := iniRead.ReadInteger(ClassName,'Top',Top);
           Width  := iniRead.ReadInteger(ClassName,'Width',Width);
           Height := iniRead.ReadInteger(ClassName,'Height',Height);
         end;
     end;
   finally
     iniRead.Free;
   end;
end;

procedure TIDSFormAssist.LayoutSave;
var
  iniOut : TINIFile;
begin
  iniOut := TINIFile.Create(sINIFile);
  try
    { Write the Left, Top, Width and Height properties to the INI file }
    with frmOwner do
      begin
        iniOut.WriteInteger(ClassName,'Left',Left);
        iniOut.WriteInteger(ClassName,'Top',Top);
        iniOut.WriteInteger(ClassName,'Width',Width);
         iniOut.WriteInteger(ClassName,'Height',Height);
      end;
    finally
      iniOut.Free;
  end;
end;

procedure TIDSFormAssist.Loaded;
begin
  inherited Loaded;
  { Ensure we are in not in design-mode, and the owner is a valid form }
  if not (csDesigning in ComponentState) then
    begin
      { Check if owner is a valid TForm }
      if (Owner = Nil) or (not (Owner is TForm)) then
          raise Exception.Create('Owner must be a TForm');
      { Retrieve the owner form }
      frmOwner := TForm(Owner);
```

```
      { Retreive the owner handle }
      if frmOwner.FormStyle = fsMDIForm then
        hwndOwner := frmOwner.ClientHandle
      else
        hwndOwner := frmOwner.Handle;
      { Restore the layout }
      if faoLayout in Options then LayoutRestore;
      { Install the event handlers }
      if faoWallpaper in Options then WallpaperInit;
    end;
end;

procedure TIDSFormAssist.SetWallpaper;
begin
  { Workaround for bitmap palette problem }
  FWallpaper.Free;
  FWallpaper := WallpaperCreate;
  { Check if a new bitmap has been set }
  if (bmpNew <> Nil) then FWallpaper.Assign(bmpNew);
  { Repaint }
  WallpaperPaint(hwndOwner);
end;

procedure TIDSFormAssist.SetWallpaperMode;
begin
  { Check if wallpaper mode has changed }
  if wmNew <> FWallpaperMode then
  begin
    { Update the wallpaper mode }
    FWallpaperMode := wmNew;
    { Repaint }
    WallpaperPaint(hwndOwner);
  end;
end;

procedure TIDSFormAssist.WallpaperChange;
var
   bmpTmp : TBitmap;
begin
   { Check if this routine has already been called }
   if blRecurs then Exit;
   { Update bitmap to avoid bitmap change bug }
   bmpTmp := TBitmap.Create;
   try
     blRecurs := True;
     bmpTmp.Assign(Wallpaper);
     { Assign new bitmap and repaint screen }
     Wallpaper := bmpTmp;
     blRecurs := False;
   finally
     bmpTmp.Free;
   end;
end;

function TIDSFormAssist.WallpaperCreate;
begin
  { Create a new bitmap object }
  Result := TBitmap.Create;
  { Set the OnChange handler }
  Result.OnChange := WallpaperChange;
end;

procedure TIDSFormAssist.WallpaperInit;
```

```
{$IFDEF VER80}
var
  dwErrCode : LongInt;
begin
  { Install the message handler }
  pPrevWndProc := TFarProc(SetWindowLong(hwndOwner,GWL_WNDPROC,
      LongInt(Addr(FormAssistWndProc))));
  if LongInt(pPrevWndProc) = 0 then
    raise Exception.Create('Unable to install message handler');
  { Start paint processing }
  blPaint := True;
end;
{$ELSE}
var
  dwErrCode : DWORD;
begin
  { Install the message handler }
  SetLastError(0);
  pPrevWndProc :=
Pointer(SetWindowLong(hwndOwner,GWL_WNDPROC,LongInt(Addr(FormAssistWndProc))));
  { Check if installation was successful }
  dwErrCode := GetLastError;
  if (LongInt(pPrevWndProc) = 0) and (dwErrCode <> 0) then
    raise Exception.Create('Unable to install message handler -
'+SysErrorMessage(dwErrCode));
  { Start paint processing }
  blPaint := True;
end;
{$ENDIF}

procedure TIDSFormAssist.WallpaperPaint;
var
  dcForm : HDC;
  rectWnd : TRect;
  iBmpW, iBmpH : Integer;
begin
  { Check whether we should repaint }
  if (not blPaint) or (Wallpaper.Empty) then Exit;
  { Retreive the device context for the window }
  dcForm := GetDC(hwndPaint);
  { Retreive the window's rectangle }
  GetWindowRect(hwndPaint,rectWnd);
  { Retreive size of wallpaper bitmap }
   with Wallpaper do
     begin
       iBmpW := Width;
       iBmpH := Height;
     end;
   { Draw the bitmap }
   if WallpaperMode = wmTile then
     WallpaperPaintTile(dcForm,rectWnd,iBmpW,iBmpH)
   else
     WallpaperPaintCentre(dcForm,rectWnd,iBmpW,iBmpH);
   { Release the device context }
   ReleaseDC(hwndPaint,dcForm);
   { Invalidate the window contents }
   InvalidateRect(hwndPaint,Nil,False);
   { Check if the user has hooked the paint event }
   if (not blRecurs) and (Assigned(FOnWallpaperPaint)) then
     begin
       blRecurs := True;
       FOnWallpaperPaint(Self);
       blRecurs := False;
```

```
   end;
end;

procedure TIDSFormAssist.WallpaperPaintTile;
var
  iX, iY : Integer;
begin
  { Start from the top row }
  iY := 0;
  { Keep drawing rows until we pass the bottom of the client rectangle }
   while iY < rectWnd.Bottom do
     begin
       { Start from the first column }
       iX := 0;
       { Keep drawing columns until we pass the right side of the client rectangle }
       while iX < rectWnd.Right do
         begin
           { Draw the bitmap at this position }
           BitBlt(dcForm,iX,iY,iBmpW,iBmpH,Wallpaper.Canvas.Handle,0,0,SRCCOPY);
           { Move to the next column }
           Inc(iX,iBmpW);
         end;
       { Move to the next row }
       Inc(iY,iBmpH);
   end;
end;

procedure TIDSFormAssist.WallpaperPaintCentre;
var
  iX, iY : Integer;
  rectFill : TRect;
begin
  { Fill the background rectangle }
  with rectFill do
    begin
      Left := 0;
      Top := 0;
      Right := rectWnd.Right - rectWnd.Left;
      Bottom := rectWnd.Bottom - rectWnd.Top;
    end;
  FillRect(dcForm,rectFill,GetStockObject(LTGRAY_BRUSH));
  { Calculate co-ordinates for draw }
  with rectWnd do
    begin
      iX := (Right - Left - iBmpW) div 2;
      iY := (Bottom - Top - iBmpH) div 2;
    end;
   { Draw the bitmap on the form }
   BitBlt(dcForm,iX, iY, iBmpW, iBmpH,Wallpaper.Canvas.Handle,0,0,SRCCOPY);
end;

{-------------------------------------------------------------}

procedure TIDSFormAssistList.Add;
begin
  { Add this new form assist object to the list }
  lstObjects.Add(faNew);
end;

constructor TIDSFormAssistList.Create;
begin
  inherited Create(cmpOwner);
  { Create internal objects }
```

```
  lstObjects := TList.Create;
end;

destructor TIDSFormAssistList.Destroy;
begin
  { Free allocated objects }
  lstObjects.Free;
  inherited Destroy;
end;

function TIDSFormAssistList.GetFormAssistFromHWND;
var
  iCount : Integer;
begin
   Result := Nil;
   { Iterate through the internal list of TIDSFormAssist objects }
   for iCount := 0 to lstObjects.Count-1 do
     { Check if this TIDSFormAssist is managing the HWND }
     if TIDSFormAssist(lstObjects[iCount]).hwndOwner = hwndSrch then
       begin
         { Return this TIDSFormAssist object }
         Result := TIDSFormAssist(lstObjects[iCount]);
         Break;
       end;
end;

procedure TIDSFormAssistList.Remove;
var
  iCount : Integer;
begin
   { Find index of this object in the list and delete it }
   for iCount := 0 to lstObjects.Count-1 do
     if lstObjects[iCount] = faOld then
       begin
         lstObjects.Delete(iCount);
         Break;
       end;
end;

initialization
begin
  { Create the form assist control list. It is freed when Application is destroyed }
  gfalControl := TIDSFormAssistList.Create(Application);
end;

end.
```

Return to Article

Return to Tips & Tricks

Return to Front Page

Finding That Caret

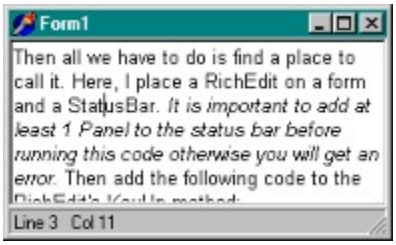
by Rolando Casumpang (r.casumpang@cgnet.com)
& Robert Vivrette - RobertV@compuserve.com

A short while back a gentleman by the name of Rolando Casumpang presented a tip on finding the caret position inside of a memo control. It was an elegant little piece of code, but he was having problems getting it to work with a RichEdit. After toying with it myself for a bit, I eventually came up with a version that worked equally well in any edit-type control. Here is the sum of our efforts:
The brains of the technique is a procedure that sends windows messages to the control in question and then fills two variables, L and C, with the Line and Column of the Caret.

```
procedure CaretPos(H: THandle; var L,C : Word);
begin
  L := SendMessage(H,EM_LINEFROMCHAR,-1,0);
  C := LoWord(SendMessage(H,EM_GETSEL,0,0)) - SendMessage(H,EM_LINEINDEX,-1,0);
end;
```

Then all we have to do is find a place to call it. Here, I place a RichEdit on a form and a StatusBar. It is important to add at least 1 Panel to the status bar before running this code otherwise you will get an error. Then add the following code to the RichEdit's KeyUp method:

```
procedure TForm1.RichEdit1KeyUp(Sender: TObject; var Key: Word; Shift: TShiftState);
var
  LineNum,ColNum : Word;
begin
  CaretPos(RichEdit1.Handle,LineNum,ColNum);
  Statusbar1.Panels[0].Text := format('Line %d   Col %d',[LineNum,ColNum]);
end;
```



If you don't want to use a StatusBar, you can just use a label control and assign the value to its Caption property.
Also, there is still one little problem with this whole technique though. The numbers get all wacky when you are selecting text. Anyone have any ideas?

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #22 - June 1997

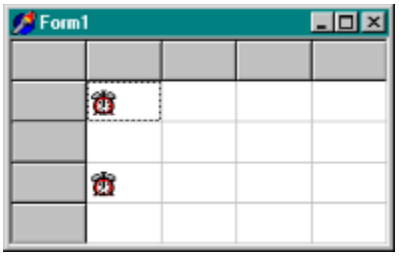
Bitmaps In String Grids

by Robert Vivrette - RobertV@compuserve.com

Last issue, a reader presented a technique to display bitmaps in string grids that was a bit more complex than it needed to be. Needless to say, I was rushed putting out the issue and did not take the time I should have to look it over. Anyway, here is a simpler solution...

All you need to do is place a StringGrid on a form and align it to alClient with the form. Then add the code shown below to its OnDrawCell, OnCreate, and OnDestroy methods. Next add the Bmp private variable to the form of type TBitmap.

Lastly, you need to put together a resource file that has a bitmap in it. I chose to use one of the button glyphs out of the IMAGES sub-directory in Delphi. The resource file contains a single bitmap named 'BMP'.



All I do to indicate where the graphic goes is to assign some special character to the cell. This is only one of many ways of accomplishing this however. You could also show the graphic based on row and/or column number for example. You also could assign the graphics into the StringGrid's Objects property. That way you could paint the text normally (with TextOut) and also have a reference to some graphic to use.
Also, one small warning. Be careful of your use of With statements in the DrawCell method. Two of the parameters passed in are Row and Col, which are also both valid properties of a StringGrid. It can produce really confusing results if a misplaced with statement tells the code to use the Row property of the StringGrid rather than the Row parameter being passed into the DrawCell function.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    procedure StringGrid1DrawCell(Sender: TObject; Col, Row: Integer;
                      Rect: TRect; State: TGridDrawState);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
  private
    Bmp : TBitmap;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.DFM}
{$R BMPS.RES}

procedure TForm1.StringGrid1DrawCell(Sender: TObject; Col, Row: Integer;
                  Rect: TRect; State: TGridDrawState);
var
  SRect,DRect : TRect;
begin
  (Sender as TStringGrid).Canvas.FillRect(Rect);
  if (Sender as TStringGrid).Cells[Row,Col] = '@' then
    begin
      SRect := Classes.Rect(0,0,Bmp.Width,Bmp.Height);
      DRect.Left := Rect.Left+3;
      DRect.Top := Rect.Top+(Rect.Bottom-Rect.Top-Bmp.Height) div 2;
      DRect.Right := DRect.Left+SRect.Right+1;
      DRect.Bottom := DRect.Top+SRect.Bottom+1;
      (Sender as TStringGrid).Canvas.BrushCopy(DRect,Bmp,SRect,clOlive);
    end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  Bmp := TBitmap.Create;
  Bmp.LoadFromResourceName(HInstance,'BMP');
  StringGrid1.Cells[1,1] := '@';
  StringGrid1.Cells[3,1] := '@';
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  Bmp.Free;
end;

end.
```

Return to Tips & Tricks

Return to Front Page

Delphi 2.0 & Delphi 3.0 on The Same Computer

by Kevin S. Gallagher - gallaghe@teleport.com

Do you need to keep Delphi 2.0 and Delphi 3.0 on your computer? Although they both can be used there is a slight problem if you want to start a project from Windows Explorer. If you double click on a project file, Delphi3 is called to execute the project. If you are using third party components that are not Delphi 3.0 ready i.e. SuccessWare's Apollo then you will see my point. I need not explain any deeper what will happen. To fix this problem until you don't need Delphi 2.0 any more simply follow the steps below:

```
Open Explorer
Select View|Options, File Type
Double Click "Delphi Project File"
Edit "Open", Press Ctrl-C, then press Cancel
Select "New" button
Type something for Action, for example  "Open With Delphi 2.0"
Press TAB, press Ctrl-V to paste from step 4
Edit path to reflect Delphi2 path
Save changes by pressing OK button.
```

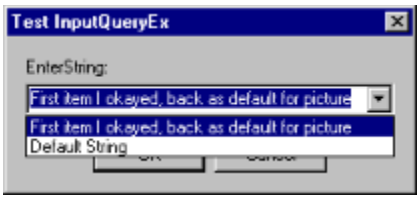
If all went OK with the steps above, you should be able to start a Delphi 3.0 project in Explorer by double clicking a project. Now to start the same project as a Delphi 2.0 project, single click the right mouse button and select the action typed in step 6, the project will be opened in Delphi 2.0.

You can expand on the above idea to include Pascal units so that you can open them with either version of Delphi, further more while you're at it, why not add an action for units that allow you to open them with Windows "notepad". Of course you can open a unit in Delphi but when opening a unit in another directory then the project directory you will need to change directories any time a new file needs to be saved into the current project directory. When I need to add code from other units I will fire up Explorer and find the needed unit, open it with notepad and copy whatever I need to the clipboard then paste to a unit in the current project. A second option is to choose (from Delphi IDE menu) File|New, text file and paste the copied code into it. Now you can use this as a personal holding area for the project.

So there you have it, working two versions of Delphi and also adding new methods of opening units.

Return to Tips & Tricks

Return to Front Page



Adding a History List to an InputQuery Box - the 32 bit version

By Gene Fowler - acorioso@ccnet.com

In UNDU issue #20, March 1997, I worked out a "template" unit for InputQueryEx, a function that is identical to InputQuery except that a ComboBox replaces the Edit so there's a history that can be reused in subsequent calls or sessions.

I made this Delphi 1 unit by pulling up the compiler directives and uses statement from dialogs.pas and put them in the inputqx unit (inputqx.pas). I made my modified declaration for the new function, and I gave old and new lines for the changes in the InputQuery function code which I had to leave for users to bring over so that I'd not be "distributing" Borland's code.

I skipped Delphi 2 because I didn't have a 32 bit operating system to run it in. But, a new system with Win 95 pre-installed became an excuse and Delphi 3 now sits on my desktop. Soooooo, a new "template" unit is in order. This will be inputqex (inputqex.pas). Note the "e" inserted. The original functions are quite different in detail and so my rewrites are, too, though anybody who chose not to wait likely didn't have any real trouble.

The pulling of code from dialogs.pas is both simpler and a tiny bit more complex. The compiler directives reduced to only {$R-}. But two uses statements, and they had extras so far as my function was concerned, so both are there but edited down. *But you will have to bring over one hidden function (GetAveCharSize) from dialogs.pas with the InputQuery code.*

Here's the "testbed" use of InputQueryEx in a Delphi 3 32 bit program.

```
procedure TFrameForm.Test1Click(Sender:TObject);
var
  ClickedOK: boolean;
begin
  { Test InputQueryEx implementation in Delphi 3 (32 bit).
    Temporary testbed for 32 bit InputQueryEx, an InputQuery
    with a history list.
    uses ..., inputqex; { inputqex.pas }
    Just above "implementation" lie my variables:
             var
               AValue: string;
               SomeValues: TStringList;
    At the end of this unit file, I have an Initialization
    section containing:
             SomeValues := TStringList.Create;
  }
  if AValue = '' then AValue := 'Default String';
  ClickedOK := InputQueryEx('Test InputQueryEx', 'EnterString:',
                            AValue, SomeValues);
end;

initialization

  SomeValues := TStringList.Create;

end.
```

Note the set-up or preparation in the note at the top of the procedure, particularly the initialization of the TStringList in the form's **initialization** section.

Source Template for InputQEx.Pas
Return to Tips & Tricks
Return to Front Page

Exceeding the Data Segment

by Robert Vivrette - RobertV@compuserve.com

A few weeks ago, I received an email from Steve Bentall at Custom Computing (slbentall@customcomputing.cix.co.uk) that covered an interesting problem and eventual solution. Here is the problem and solution in his words…

The Problem

The application I have been developing has hit the 'Compiler Error 49 Data Segment Too Large' brick wall and I am desperate to come up with an approach that will allow me to complete the development. The app currently consists of 95 Forms in 97 units and provides approximately 75% of it's required functionality at this stage. It is written in D1 and as much as I would love to move to D3, this app will be running under w3.x for at least 18 months after initial release.

I have spent some time looking closely at the app using the Turbo Debugger and am staggered at how much of the Data Segment appears to be cluttered with VCL related data for components that I am not aware that I require.

I am willing to consider almost any approach (except moving from D1 unfortunately) and wondered whether you could suggest what I should try or where I may find guidelines and recommendations (Moving more literals into Resource Strings?, splitting certain functionality into DLL(s)? etc?).

Later… The Solution!

I have made quite remarkable progress more by luck than by judgment. I have managed to reduce the size of the Data Segment (excluding Stack and Local Heap) from 42,838 bytes to 11,894 bytes.

I realized almost by chance that variables declared in a unit outside of any procedures or functions are placed in the Data Segment even if they are local to the unit by virtue of the fact that they are declared in the implementation section. I had incorrectly assumed that only those declared in the interface section would be placed in the Data Segment.

Moving these variables out of the Data Segment was simply a case of declaring them as Private variables of the unit's form.

Variables declared outside of any procedures or functions in the implementation section are only accessible within the unit and are not visible in the MAP as Public Variables hence I didn't spot them. I assume that would explain why I am having difficulties identifying where the remaining space between the Public Variables is defined.

A cautionary tale for others perhaps?

Return to Tips & Tricks

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #22 - June 1997

A Review of Addict by Addictive Software

by Robert Vivrette - RobertV@compuserve.com

Professional touches to an application really set it apart from the crowd. What better way to set apart your application than by giving it the ability to do Spell Checking as well as access to a Thesaurus? It isn't as difficult as you might think!

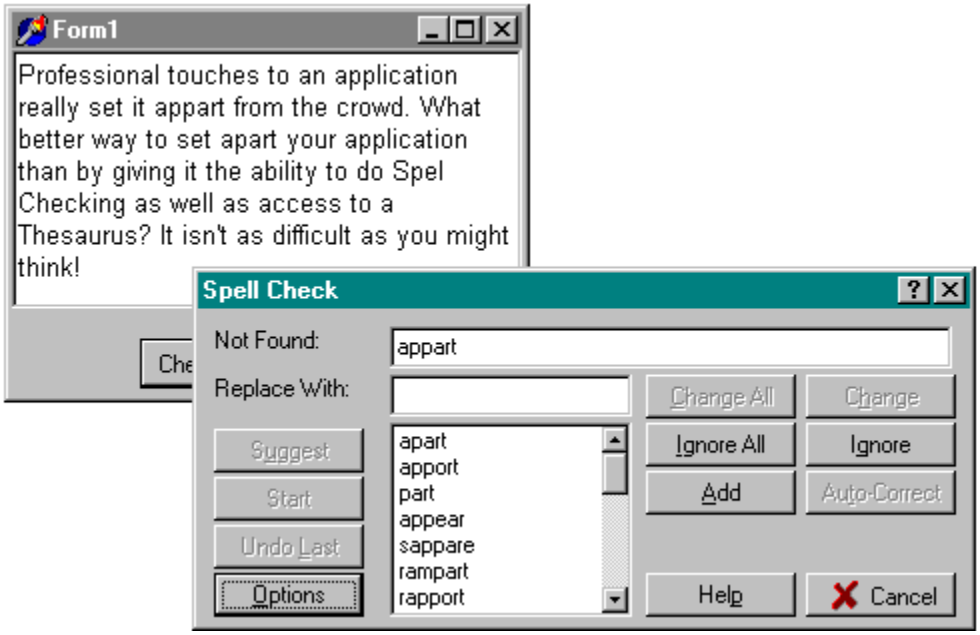
To show you just how easy it is, lets build a sample application using Addictive Software's AddictSpell component. Don't blink or you will miss it!

Start a new project. Add an RichEdit control, a Button, and a AddictSpell component. It should look something like this:

Double-click on the button and add the following code:

```
procedure TForm1.BitBtn1Click(Sender: TObject);
begin
  AddictSpell1.Check(RichEdit1);
end;
```

Now compile and run it! (I told you not to blink…) I type in some text and click on the button, and viola! Addict goes to work identifying spelling errors in the RichEdit control.



This is the most basic of examples, but I think you can see just how easy it is to add a very powerful feature to your applications.

Addict supports user-dictionaries as well as the Microsoft Word dictionaries (that will help save disk space!). If you have a list of custom words you need to add, there is a Dictionary compiler to process them for you automatically, and you can setup multiple user-dictionaries and have any or all of them

active at the same time. It automatically understands how to spell check any descendent of TCustomEdit, TCustomMemo, and TCustomRichEdit and a number of 3rd party edit controls. It also can check Pchars and strings. Addict also has a large number of configuration options including the ability to allow it to handle HTML tags, and spell check in the background.
For fast and simple spell checking, you need look no further than AddictSpell. It is fast and incredibly easy to use!

For a trial version, you can visit their web site at http://www.flinthills.com/~addict. You can also contact Addictive Software at addict@flinthills.com. The full version includes source code.

By the way… the Thesaurus component is great too… A review of that will have to wait until a future issue of UNDU… Stay tuned!

Return to Front Page

The Unofficial Newsletter of Delphi Users - Issue #22 - June 1997

A Review of the Programmers Guild Animated Tray Icon

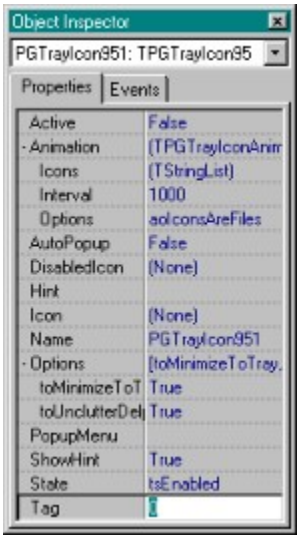
by Robert Vivrette - RobertV@compuserve.com

Windows 95 adds all sorts of handy user interface features that Delphi programmers love to play with. One of the handiest is the ability to insert an icon in the Windows 95 TaskBar Tray area and use it to control various features of an application.

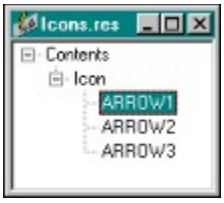
Programmers Guild has offered their Animated Tray Icon for some time now and I have been using it for the last 6 months or so (in various incarnations). Recently they released an updated version that now supports all versions of Delphi as well as C++ Builder. This latest version does not disappoint! Since the first time I used it I was hooked!

To show you how simple it is, I am going to walk through a short illustration of its use:

After a ridiculously simple install (it even puts it into Delphi for you), I am ready to go. I start a new project and plop a PGTrayIcon95 onto the form. Here is a look at some of the properties you have at your disposal:



Let's make an animated tray icon… After placing the PGTrayIcon95 component on to the form, I set its **State** property to **tsAnimated**. Now since this is going to be an animation, we need to give it something to animate. I create a RES file (with Delphi's ImageEditor) and add 3 icon resources to it. I make the icons 16x16 and in 16 colors, and paint 3 different arrows on their surfaces. I then save the RES file as ICONS.RES. The resource file looks something like this:



Next, I provide the names of the 3 icon resources in the **Animation.Icons** stringlist. All I need to do here is type in their names: ARROW1, ARROW2, and ARROW3. Next I set its **Animation.Options** property to **aoIconsInExe**. This tells the component that the images are in a resource bound in with the EXE.
Two steps left! I need to tell Delphi to bind in the resource file so I include **{$R ICONS.RES}** somewhere in the Form's unit file and then lastly, I set the components **Active** property to **True**. Compile and Run!

As you can see when the program is running, the little icon is down in the TaskTray area flipping around between the three graphics. Pretty cool! And I didn't even have to write any code. Of course it really doesn't do anything other than animate, so let's add a popup menu.

I add a TPopupMenu to the form and add 2 menu items to it: Show Menu and Hide Menu. Here are the OnClick events:

```
procedure TForm1.HideForm1Click(Sender: TObject);
begin
  Hide;
end;

procedure TForm1.ShowForm1Click(Sender: TObject);
begin
  Show;
end;
```

And then I need to tell the PGTrayIcon95 component that it has a popup menu by connecting its PopupMenu property to my newly added PopupMenu1 component. Last step is to set the PGTrayIcon95 components AutoPopup property to True. Compile and run it again…

Now I get the same thing, but when I right-click on the dancing arrow, I get a popup menu allowing me to selectively hide or show the form.

This has been a pretty simple example of how the PGTrayIcon95 works, and I haven't even touched most of its capabilities. Some of its added features include the ability to set a flyover hint for the icon, showing the icon in an enabled or disabled state, as well as a full host of user events such as OnClick, OnDblClick, OnMouseMove, OnMouseEnter, OnMouseExit, OnAppMinimize, OnAppRestore, and OnStateChanging - **Whew!!!**
You can get a trial version of the component (and others) at Programmers Guild's Web site http://www.programmersguild.com. Registering the component will cost you a mere $10 and $25 will get you the source code. A bargain!

I love PGTrayIcon95 (except for the name perhaps) and would highly recommend it to anyone who wants an easy and effective way to manage icons in the TaskTray area.

You can reach Programmers Guild at feedback@programmersguild.com or by snail mail at:

```
The Programmers' Guild
7 Main Street
St. George, N.B.
Canada
E0G 2Y0
```

Return to Front Page

Using the Win95 Registry

by Wim Vandersmissen - pin10012@pophost.ping.be

After playing around with Delphi 2 for a few months, I wanted to use the Win 95 registry instead of the old INI files to store the options used by my programs. After browsing the Delphi and Windows 32 API help files, I concluded that the TRegistry object would be the best to do this. Now the only thing that was left was to find out how you had to use the methods of this object, 'cause there are no examples. That's why I decided to write this article, so that you don't have to find out the hard way (like me and many others) ;-) The first thing to do when your form uses the registry is to add the **Registry** unit to the uses clause at the top of your source-code. In 90% of the cases you want to set some properties of your form to predefined settings you store in the registry (e.g. the width and height of the form), so you'll have to add some code to the OnCreate-event of your form too.

Now let's have a look at an example. The following program will check at startup if a certain key already exists in the registry, if so, it will read a value from it, if not, the key will be created and set to a certain default value.

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs, Registry, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    CountLabel: TLabel;
    Label2: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
 Close;
end;
```

```
procedure TForm1.FormCreate(Sender: TObject);
var
 theReg  : TRegistry;
 KeyName : String;
 Counter : Integer;
begin
 theReg := TRegistry.Create;
 KeyName := 'Software\Wim\Test';
 if (theReg.KeyExists(KeyName)) then
   begin
     theReg.OpenKey(KeyName, False);
     Counter := theReg.ReadInteger('UsageCount');
   end
 else
   begin
     theReg.OpenKey(KeyName, True);
     Counter := 0;
   end;
 Counter := Counter + 1;
 theReg.WriteInteger('UsageCount', Counter);
 CountLabel.Caption := IntToStr(Counter);
 theReg.Free;
end;

end.
```

Now let me try to explain how it works. First you create an instance of the TRegistry object, using TRegistry.Create. Don't forget to do this, because if you don't, strange things can happen (in the good ol' days it would be a GPF, but now. who knows). Also, when you're done reading from or writing to the registry, don't forget to free the memory used by the TRegistry object, by using the Free method.

Now that we have a valid instance of the TRegistry object, I check whether or not the key 'Software\Wim\Test' already exists. To do this, you just call the KeyExists method, using the keyname as a parameter. The function returns a True when the key already exists and False if it doesn't. I use a relative key, but you can also use an absolute key, by starting the name with a backslash (using '\HKEY_CURRENT_USER\Software\Wim\Test' would give the same result).

Let's suppose the key doesn't exist yet. The first thing you have to then is to create and open the key. You can do this by calling the OpenKey method. This method needs 2 parameters which are the key's name and a boolean value. To create the key you have to call OpenKey(<key name>, True) . After this call you can add all kinds of data values to this key. I just add the UsageCount as an Integer, by using the WriteInteger method.

If the key already exists, I open it Using OpenKey, but this time the boolean value is set to False, because the key already exists (you could set it to true though. doesn't matter in this example). After this, I read the value stored in the UsageCount data value using the ReadInteger method.

As you can see, it isn't all that difficult to use the registry. If you'd like to get more information are have problems using the registry, you can always let me know of course.

May the source be with you!

Return to Tips & Tricks

Return to Front Page

Word Search Tip

by Arlan Mock - amock@alum.mit.edu

When you bring up the find/replace dialog box in Delphi, do you ever wish the editor would automatically enter the word your cursor is currently on? In Delphi 1, select Options|Environment. On the Editor Options tab check the 'Find text at cursor' checkbox. In Delphi 2, select Tools|Options. You'll find the same checkbox on the Editor tab.

Return to Tips & Tricks
Return to Front Page

Unbridled Acceleration

by Arlan Mock - amock@alum.mit.edu

I have come across a strange bug in Delphi having to do with accelerator keys used in the Caption property of buttons, checkboxes, labels with a FocusControl, etc. Normally, to use an accelerator key you must press the Alt-<char> combination. However, under certain conditions, the accelerator is executed without the use of the Alt key. This can be reproduced in both Delphi 1 and Delphi 2.

0 to 60 WITHOUT THE ALT KEY

To see this behavior, start a new project and place two buttons on the form. Set the captions of the buttons to '&A Button' and '&B Button'. Also, assign an OnClick event handler for each button. For example, for the A button, you could use ShowMessage('You pressed the A button'). Likewise, have the 'B' button show a message that you have pressed the 'B' button. Now, run the project. Click on the 'A' button so that it receives focus and displays its message. Close the message box, and then, using the keyboard, press the 'B' key without the Alt key while button A has focus. The message dialog for the 'B' button will be erroneously displayed.

It seems that an accelerator key executes without the Alt key whenever a TButton, a TStringGrid with Options.goEditing = false or a TCheckbox has focus.

UNDER THE HOOD

This section discusses the gory details behind the problem. If you just want to know how to correct this behavior, skip to the section entitled 'THROTTLING BACK A TAD'.

It seems any TWinControl descendent that doesn't want to accept characters and that has focus will cause accelerator keys to work without the Alt key. To be more specific, this behavior appears while focus is held by any TWinControl descendent that does not set the DLGC_WANTCHARS flag in response to the WM_GETDLGCODE message. The WM_GETDLGCODE message is sent from the TWinControl.CNChar message handler. After the call to the WMGETDLGCODE message handler returns, CNChar checks the DLGC_WANTCHARS flag. If it is not set, CNChar tells the parent form to perform the CM_DIALOGCHAR message.

It is in the CMDIALOGCHAR message handlers that the problem starts to surface. CM_DIALOGCHAR is the internal Delphi message that is overridden if a control can make use of an accelerator key. TForm inherits its CMDIALOGCHAR message handler from TWinControl. This message handler is different than other such message handlers in that it does not check for accelerator keys. Instead it broadcasts the CM_DIALOGCHAR message to all child controls to see if the children have an accelerator key. So when TForm performs this message, it broadcasts the message to all its child controls. Any container controls on the form, such as panels, will in turn broadcast the message to all its child controls. This continues until a control has a CMDIALOGCHAR message handler that checks for an accelerator key that matches the character key that was pressed or until all controls have had a chance to handle the message, none of which have a corresponding accelerator.

All CMDIALOGCHAR message handlers in the VCL other than TWinControl's check for an accelerator key. Each one does this by calling a function in FORMS.PAS called IsAccel. This function requires two parameters - the key that was pressed and the caption. IsAccel checks to see if the key matches the character after the caption's first '&' character. Unfortunately, IsAccel does not have a parameter for the

state of the control, alt and shift keys!

My initial thought on solving this problem was to modify IsAccel in FORMS.PAS to have a third parameter for the key state. This doesn't work well because that changes the interface section of the FORMS unit, and any other unit that uses FORMS would need to be recompiled. This would require source code for any such unit, which may not be available. *This is one reason it is a good idea to never modify the interface section of any VCL source code.*

THROTTLING BACK A TAD

The fix I use involves finding and modifying each CMDialogChar message handler in the VCL that calls the IsAccel procedure in FORMS.PAS. *If you decide to do this, be sure to back up the old source, just in case.* Here's an example of the modification in TButton.CMDIALOGCHAR. It makes use of the KeyData field of the Message parameter, which contains information about the state of Ctrl, Alt and Shift. The modification is underlined and marked by **{>}**...**{<}**.

```
procedure TButton.CMDialogChar(var Message: TCMDialogChar);
begin
  with Message do
    if {>} (KeyDataToShiftState(KeyData) = [ssAlt]) and {<}
      IsAccel(CharCode, Caption) and CanFocus then
      begin
        Click;
        Result := 1;
      end
    else
      inherited;
end;
```

To find all such routines in the VCL, just search for IsAccel in BUTTONS.PAS, STDCTRLS.PAS, TABNOTBK.PAS and TABS.PAS. Not all CMDialogChar message handlers have a 'with Message do' statement. In such cases, you will have to use Message.KeyData instead of KeyData.

FOREIGN VEHICLES

One last note - any third party components you have that override CMDialogChar may need to be modified as well.

Editors Note:

*I just want to reinforce to UNDU readers that this solution involves making some minor changes to the VCL source code. You should **definitely** backup any files prior to modifying them.*

Return to Tips & Tricks

Return to Front Page

Delphi Users Groups

A while back, I sent out emails to all the Delphi Users Groups I could get my hands on and asked them to send me some basic information about their group. This issue, I will be starting my attempt at publishing this material. I only got a few responses back, but hopefully it will grow with time. As a result, it will be a bit sparse this time, but more information will definitely be added as we go.

If you run a Delphi Users Group that is not on this list, please feel free to send me an email with the essentials. Please include basic information about where and when the group meets, and particularly any web links you may have to support the group. I am going to keep the info to a minimum so I can include in each issue (or maybe every other issue). I have provided a sample form to help you provide the right information.

California

Orange County Delphi Users Group

North Bay Delphi SIG

Canada

Metro Halifax Delphi Developers Group

Iowa

Central Iowa Delphi Users Group

Michigan

Royal Oak Delphi Users Group

North Carolina

Research Triangle Park Delphi Users Group

Pennsylvania

Pittsburgh Area Delphi Users Group

Return to Front Page

Group Name:	Orange County Delphi Users Group
Person(s) In Charge:	Maurie Seymour
Meeting Date/Time:	Second Saturday of each month at 1:00pm
Meeting Location:	1520 Brookhollow Drive, Suite 38, Santa Ana, CA
Dues:	???
Email Address:	delphibegin.sig@ocipug.org or mauries@pacbell.net
Phone Number:	(714) 633-2914
Internet URL:	http://www.ocipug.org
Description:	

Return to Delphi Users Groups

Group Name:	Pittsburgh Area Delphi Users Group
Person(s) In Charge:	Kirk A. Farra
Meeting Date/Time:	Second Thursday of the month at 7:00pm
Meeting Location:	Mastech Inc.
Dues:	None
Email Address:	
Phone Number:	(412) 452-6121
Internet URL:	http://www.nauticom.net/www/kfarra
Description:	We cover Delphi and any related products/tools. Members are exposed to many 3rd party components and have a local support network for Q/A. Members get to review many products before they are released and get information on local job opportunities and contracting positions.

Return to Delphi Users Groups

Group Name:	Central Iowa Delphi Users Group
Person(s) In Charge:	David W. Body
Meeting Date/Time:	Third Thursday of every month at 5:30pm
Meeting Location:	Offices of Iowa Bankers Association, Liberty Building, 418 6th Ave. Suite 430, Des Moines, Iowa.
Dues:	None
Email Address:	
Phone Number:	(515) 984-6243
Internet URL:	http://www.bigcreek.com/delphi
Description:	

Return to Delphi Users Groups

Group Name:	Royal Oak Delphi Users Group
Person(s) In Charge:	???
Meeting Date/Time:	First Thursday of each month at 7:00pm
Meeting Location:	William Beaumont Hospital (Lower Level), Royal Oak, MI. Classrooms in AB-West building.
Dues:	None
Email Address:	???
Phone Number:	(810) 551-5000
Internet URL:	http://ourworld.compuserve.com/homepages/STNICOL/
Description:	Constructing DLL's, SQL Server 6.5, CGI, Email enabled databases, data collection alternatives, VCL parties, book review night, etc.

Return to Delphi Users Groups

Group Name:	Metro Halifax Delphi Developers Group
Person(s) In Charge:	Dave Hackett
Meeting Date/Time:	First Tuesday of each month from 7:00pm to 9:00pm
Meeting Location:	Maritime Life Business Park
Dues:	None
Email Address:	71650.2646@compuserve.com
Phone Number:	
Internet URL:	
Description:	News, Q&A, Presentations

Return to Delphi Users Groups

Group Name: North Bay Delphi Special Interest Group
Person(s) In Charge: Bjarne Winkler
Meeting Date/Time: Second and Fourth Wednesday of each month from 7:00pm to 9:00pm
Meeting Location:
Dues:
Email Address: nts4618086@aol.com
Phone Number:
Internet URL:
Description: Announcements, Q&A, Special topics each meeting.

Return to Delphi Users Groups

Group Name:	Research Triangle Park Delphi Users Group
Person(s) In Charge:	Mark Hutchinson
Meeting Date/Time:	Third Wednesday of each month, 5pm
Meeting Location:	New Horizons Computer Learning Center - Willow Oak Bldg, Suite 116, 4625 Creekstone Drive, Durham, NC
Dues:	
Email Address:	aikimark@aol.com
Phone Number:	
Internet URL:	http://www.geocities.com/CapeCanaveral/Lab/1185/rtpdig.html
Description:	

Return to Delphi Users Groups

Form For Users Groups

If your Delphi Users Group is not mentioned in the list, please take a moment to email me the particulars about your group. To make the information as uniform as possible, please *cut and paste this form* into an email and send it to me at RobertV@compuserve.com. Please only send one if you are in charge of the group… I don't want to have to wade through 30-40 copies for each group… Thanks!

Group Name:

Person(s) In Charge:

Meeting Date/Time:

Meeting Location (include City & State):

Dues:

Email Address:

Phone Number:

Internet URL:

Brief description of meeting format:

Return to Delphi Users Groups

Return to Front Page

Getting Control of the Control Panel

by Andrew Jeffries - adjeffries@clara.net

An application I have just written has about thirty settings which are loaded at start-up, so I thought it would be useful to have these in a separate application, or at least in a part of the system that can be accessed separately from the main application. Then my thoughts turned to writing a Control Panel Application (or Applet as Microsoft likes them known).
After delving through then Windows API help files (16 & 32-bit) and a lot of conversion from horrid C notation to our beloved Delphi I have now managed that very task…and was surprised how easy it works.

Applet Basics Covered (As easy as A..B..C)

A Control Panel Applet is a DLL (Dynamically-Linked Library) renamed to a CPL. If you can write a DLL you are not far behind writing a Control Panel Applet. If you can write a Delphi application you are not far behind writing a DLL.

So to make our CPL, all we need is a small application that is working fine. For our example, this will simply save the name of a user in an INI file. Yes, I know we should use the registry (for 32-bit applications) but lets walk before we can run.

We will then convert it into a DLL (available to our main application) and then into a CPL. The example doesnt do a lot you understand but it demonstrates the principles nicely. The following example is for 16-bit applets. If you wish to create 32-bit applets read the note at the end of this article.

User Name Application

The first step is to write a working application. The example shown below is the design of the form, nothing fancy just a label, an edit box and a button.

In the OnCreate event for the form put the following code:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
  with TIniFile.Create('c:\windows\myapp.ini') do
    begin
      Edit1.Text := ReadString('User Details', 'Name', '(None)');
      Free;
    end;
end;
```

With IniFiles in the forms uses clause, this will open (if it doesnt exist - create) a file called myapp.ini in the C:\Windows folder. In the OnClick event for the TButton put the following code:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with TIniFile.Create('c:\windows\myapp.ini') do
    begin
      WriteString('User Details', 'Name', Edit1.Text);
      Free;
    end;
  Close;
end;
```

And that is our application written. All it does is opens the INI file, creating one if necessary, reads in the Users name and saves any changes out upon clicking the OK button. This could be made more efficient, but this will do for our example.

Recipe for a DLL

Now our application is working, it is time to convert it to a DLL. This is very simple. The project source for the application should currently look something like this:

```
program Project1;

uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1};

{$R *.RES}

begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

To convert this to a DLL all we need to do is alter the initial declaration so that Delphi knows to compile it as a DLL and export a routine that does the startup. It could be changed to the following:

```
library Project1;

uses
  Forms,
  Unit1 in 'UNIT1.PAS' {Form1};

{$R *.RES}

procedure RunApp; export;
begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end;

exports
  RunApp;

begin
end.
```

All we have done is change the header from **program** to **library**, move the routine that opens the form and runs the application to its own procedure and exported that function. This is now a fully functional DLL and could be used by a larger application. But our task is to create a Control Panel Applet, so…

Making the Final Leap

This is where all our work is done and this is where the source code starts getting slightly complex. Before we start altering our source code, a small application of how the Control Panel Application (CONTROL.EXE) talks to our Control Panel Applet (MYAPPLET.CPL) is in order.

Control Panel uses a Callback function to start up the Applet and gain information about it. Now technically, many Applets can be contained within a single CPL file although for the moment we shall only be concerned with doing one for the moment. Now all this callback business sounds a bit complex, but it isnt.

In this context, all it means is that you have to declare a function with a standard set of parameters. Control Panel will then execute this function passing what it wants your Applet to do and you take care of

the information it passes. Does this still sound complicated? Here is a small (simplified) example:

You have a function called: function CPLApplet (CPLMessage):Result;

Control Panel runs this function sending the message Are you a Control Panel applet? - to which well reply true as the result.

Control Panel then runs the function again sending the message How many Icons do you want in my Control Panel Window? - To which well reply 1 as the result.

Control Panel then sends the message OK, give me the information for your first applet - and you return the a handle to an icon, and short and long names (for under the icon and the bottom of the window respectively). This message will be repeated for each of the Applets required, although we already stated we only want one.

Control Panel then sends a message when the user double clicks on the icon in Control Panel.

Now, does that seem a bit easier? Right, now we start delving into code and actually convert our DLL in to a CPL.

The only real thing to do is to declare a function called CPLApplet with the parameters Control Panel expects (slightly more complex than the example above!). The following function does all the talking with Control Panel. In the comments CP represents the Control Panel application (CONTROL.EXE).

```
function CPLApplet (hWndCpl: HWnd; msg: Word; lParam1: Longint;
            var NewCPLInfo: TNewCPLInfo): Longint; export;
begin
 case msg of
  CPL_INIT: CPlApplet := 1;
   {CP asks Are you an Applet  We reply 1 - Yes I am}

  CPL_GETCOUNT: CPlApplet := 1;
   {CP asks How many icons do you want  We reply 1 - One icon,
    please}

  CPL_NEWINQUIRE:
   {CP sends this message once for every icon you require. In our
    case this is only sent once, so we dont need to concern ourself
    with what applet number CP wants to know about}
   begin
    with NewCPLInfo do begin
     dwSize:=sizeof(TNewCPLInfo);
     dwFlags := 0;
     dwHelpContext := 0;
     lData := 0;
     szHelpFile[0]:= #0;
     {Now comes the intersting bit; our icon and names}
     Icon :=Application.Icon.Handle;
     StrPCopy(szName,'My First Applet');
     StrPCopy(szInfo,'My first Control Panel Applet (in Delphi!)');
    end;
   end;

  CPL_DBLCLK: RunApp;
 end;
end;
```

Now we nearly have a functional Control Panel Applet. Only five small steps left before it actually works from Control Panel.

Add the following libraries to the uses clause of the DPR - WinProcs, WinTypes, CPL, SysUtils.

Add the CPLApplet function to the exports clause of the DPR.

Compile the project.

Rename the resulting DLL to a CPL, for example MYAPPLET.CPL.

Copy this CPL file to your Windows\System folder\directory.

Now the next time you run Control Panel, your applet will appear…

32-Bit Considerations

For an applet to be compiled under Delphi 2.0 a number of other considerations are necessary, for example there is no defined library CPL.PAS that was included with Delphi 1.0, so the constants and types need redeclaring as shown below. This will now work under Windows 95 but further thoughts are necessary for it to run under Windows NT. As I am new to programming for NT, any thoughts on this issue, e-mailed to me, would be greatly appreciated.

```
// Control Panel Declarations
type
  TNewCPLInfo = record
    dwSize: Longint;
    dwFlags: Longint;
    dwHelpContext: Longint;
    lData: Longint;
    Icon: HIcon;
    szName: array[0..31] of Char;
    szInfo: array[0..63] of Char;
    szHelpFile: array[0..127] of Char;
  end;

const
  CPL_INIT         = 1;
  CPL_GETCOUNT     = 2;
  CPL_DBLCLK       = 5;
  CPL_NEWINQUIRE   = 8;
```

Source Code for Main Form (MAINFORM.PAS)

Source Code for Project File (MYAPPLET.DPR)

Form Definition File (MAINFORM.DFM)

Return to Front Page

Source Code for MainForm.PAS

```
unit Mainform;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, IniFiles;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  with TIniFile.Create('c:\windows\myapp.ini') do
    begin
      Edit1.Text:=ReadString('User Details', 'Name', '(None)');
      Free;
    end;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
  Close;
end;

end.
```

Return to Article

Return to Front Page

Form Definition File - MainForm.DFM

```
object Form1: TForm1
  Left = 241
  Top = 134
  Width = 246
  Height = 126
  Caption = 'Our Application'
  Font.Color = clWindowText
  Font.Height = -13
  Font.Name = 'System'
  Font.Style = []
  PixelsPerInch = 96
  OnCreate = FormCreate
  TextHeight = 16
  object Label1: TLabel
    Left = 12
    Top = 12
    Width = 75
    Height = 16
    Caption = 'User name:'
  end
  object Edit1: TEdit
    Left = 96
    Top = 8
    Width = 121
    Height = 24
    TabOrder = 0
    Text = 'Edit1'
  end
  object Button1: TButton
    Left = 60
    Top = 52
    Width = 89
    Height = 33
    Caption = '&OK'
    Default = True
    TabOrder = 1
    OnClick = Button1Click
  end
end
```

Return to Article

Return to Front Page

Control Panel Project Source

```
library Myapplet;

uses
  Forms,
  Mainform in 'MAINFORM.PAS', {Form1}
  WinTypes, WinProcs, SysUtils;


{$R *.RES}

procedure RunApp; export;
begin
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end;

function CPLApplet (hWndCpl: HWnd; msg: Word; lParam1: Longint;
                var NewCPLInfo: TNewCPLInfo): Longint; export;
begin
 case msg of
  CPL_INIT: begin
    CPlApplet:=1;
   end;
   {CP asks "Are you an Applet"  We reply 1 - "Yes I am"}

  CPL_GETCOUNT:begin
   CPlApplet:=1;
  end;
   {CP asks "How many icons do you want"  We reply 1 - "One icon,
    please"}

  CPL_NEWINQUIRE:
   {CP sends this message once for every icon you require. In our
    case this is only sent once, so we dont need to concern ourself
    with what applet number CP wants to know about}
   begin
    with NewCPLInfo do begin
     dwSize:=sizeof(TNewCPLInfo);
     dwFlags := 0;
     dwHelpContext := 0;
     lData := 0;
     szHelpFile[0]:= #0;
     {Now comes the intersting bit; our icon and names}
     Icon :=Application.Icon.Handle;
     StrPCopy(szName,'My First Applet');
     StrPCopy(szInfo,'My first Control Panel Applet (in Delphi!)');
    end;
   end;

  CPL_DBLCLK: RunApp;

 end;
end;


exports
  RunApp,
  CPLApplet;

begin
```

```
end.
```

Return to Article

Return to Front Page

Source Template for InputQuery

```
unit InputQEx;  { inputqex.PAS, for Delphi 32 bit }

{ This unit is a template in which you can build InputQueryEx, a
  function identical to InputQuery but with a ComboBox in place of
  the Edit so that you have a history to reuse in subsequent calls
  or sessions.

  You must bring code for two functions in from dialogs.pas (in
  the Delphi 2/3 VCL source code). I have only Delphi 3, but assume
  this code was likely the same in Delphi 2.

  Bring in GetAveCharSize from the top of the implementation section
  and put it where I've made the comment below. Then, bring in the
  code for InputQuery BUT WITHOUT THE DECLARATION. Start it with
  the VAR(iables) list for the function and, then, the code. The
  first change will be a couple entries in that VAR list.

                                                    --Gene fowler
                                                      May, 1997
}

{$R-}
interface

uses Windows, Classes, Graphics, Controls, Forms, StdCtrls, Dialogs;

function InputQueryEx(const ACaption, APrompt: string;
  var Value: string; var Values: TStringList): Boolean;

implementation

uses Consts;

{function GetAveCharSize(Canvas: TCanvas): TPoint;
 goes here. It's under implementation in dialogs.pas}


function InputQueryEx(const ACaption, APrompt: string;
  var Value: string; var Values: TStringList): Boolean;
{ Copy the var(iables)list here}

{ Add these two items to the var list }
  i: integer;
  S: string;

{ Insert from begin here. Then, in that whole code block
  find the lines that match one of my old-lines blocks and
  replace it with the new-lines block beside that.}

{First block}

{old} Edit := TEdit.Create(Form);
{old} with Edit do
{old} begin
{old}   Parent := Form;
{old}   Left := Prompt.Left;
{old}   Top := MulDiv(19, DialogUnits.Y, 8);
{old}   Width := MulDiv(164, DialogUnits.X, 4);
{old}   MaxLength := 255;
{old}   Text := Value;
```

```
{old}   SelectAll;
{old} end;

{new} Combo := TComboBox.Create(Form);
{new} with Combo do
{new}   begin
{new}     Parent := Form;
{new}     Left := Prompt.Left;
{new}     Top := MulDiv(19, DialogUnits.Y, 8);
{new}     Width := MulDiv(164, DialogUnits.X, 4);
{new}     MaxLength := 127;
{new}     SelectAll;
{new}     Text := Value;
{new}     Combo.Items.Clear;
{new}     if Values.Count > 0 then
{new}       begin
{new}         For i := 0 to Values.Count - 1 do
{new}           begin
{new}             S := Values[i];
{new}             Combo.Items.Add(S);
{new}           end;
{new}         if Combo.Items[0]  Combo.text then
{new}           begin
{new}             Combo.Items.Insert(0, Combo.Text);
{new}             for i := 0 to Combo.Items.Count - 1 do
{new}               if Combo.Items[i] = Combo.Items[0] then
{new}                 Combo.Items.Delete(i);
{new}           end
{new}       end
{new}     else
{new}       Combo.Items.Add(Combo.Text)
{new}   end;
```

{Second block}

```
{old} if Form.ShowModal = mrOk then
{old}   begin
{old}     Value := Edit.Text;
{old}     Result := True;
{old}   end;

{new} if Form.ShowModal = mrOK then
{new}   begin
{new}     Result := True;
{new}     Value  := Combo.Text;
{new}     if Combo.Items[0]  Combo.Text then
{new}       Combo.Items.Insert(0, Combo.Text);
{new}     for i := 1 to Combo.Items.Count - 1 do
{new}       if Combo.Items[i] = Combo.Items[0] then
{new}         Combo.Items.Delete(i);
{new}     Values.Clear;
{new}     For i := 0 to Combo.Items.Count - 1 do
{new}       begin
{new}         S := Combo.Items[i];
{new}         Values.Add(S);
{new}       end;
{new}   end;
```

Return to Article

Return to Tips & Tricks

Return to Front Page