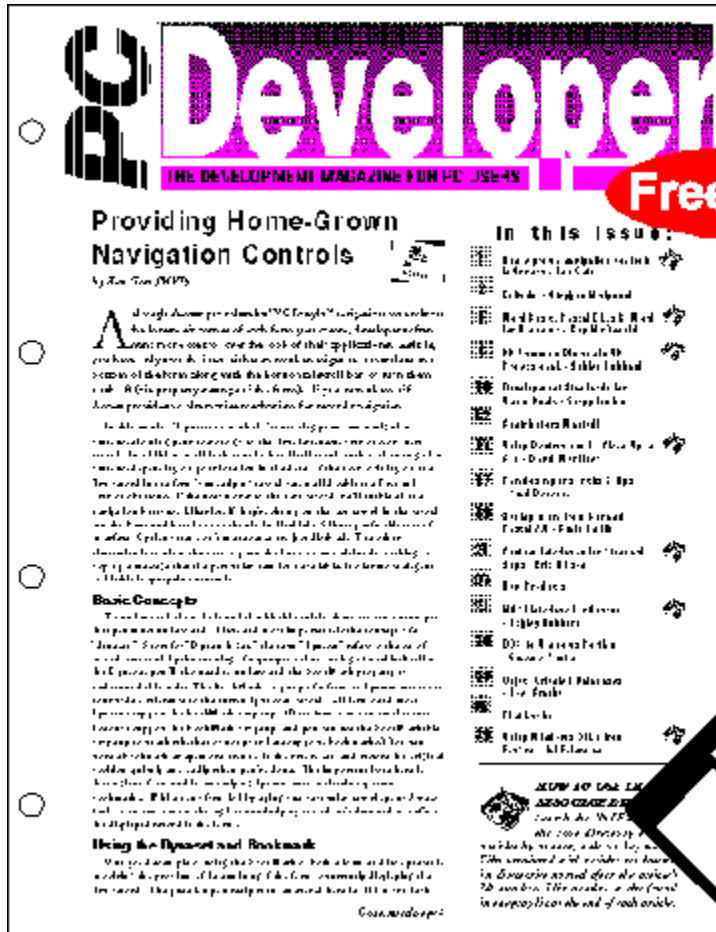


PC Developer magazine - click anywhere to explore



Developer

THE DEVELOPMENT MAGAZINE FOR PC USERS

Providing Home-Grown Navigation Controls

by Tom Van Dyke

Although homegrown controls "look rough" compared to modern Windows controls, they can give your users a unique experience. This article shows you how to create a set of homegrown controls with the look and feel of Windows controls. It also provides information on how to use the Windows API to create a set of homegrown controls that look like Windows controls.

Basic Concepts

The basic concepts of homegrown controls are: 1. Use the Windows API to create a set of homegrown controls that look like Windows controls. 2. Use the Windows API to create a set of homegrown controls that look like Windows controls. 3. Use the Windows API to create a set of homegrown controls that look like Windows controls.

Using the Dynamic and Static

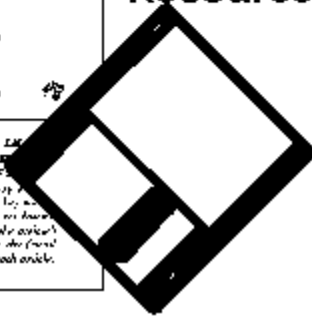
Dynamic and static controls are used to create a set of homegrown controls that look like Windows controls. Dynamic controls are used to create a set of homegrown controls that look like Windows controls. Static controls are used to create a set of homegrown controls that look like Windows controls.

Copyright © 1995 by Tom Van Dyke

Free Issue!

- ### In this Issue:
- Articles
 - Authors
 - Contributing
 - Sample 1
 - Sample 2
 - Sample 3
 - Sample 4
 - Index S/W
 - Advertising
 - Resource disks

Articles
Authors
Contributing
Sample 1
Sample 2
Sample 3
Sample 4
Index S/W
Advertising
Resource disks



Promotional Help File Version 2.0

SUBSCRIPTION INFORMATION

SUBSCRIBE

PC Developer magazine is easy to read and understand, yet tackles tough language specific and generic issues. In depth articles look at maximizing your performance with Access, Visual Basic, Delphi, C/C++, Pascal, Paradox, FoxPro, CA-Visual Objects and more. Indexed Resource diskettes let you play with source code and sample executables. PC Developer now has over 70 pages packed with information to keep you up with the best. Sharpen your skills and make your code more efficient by learning the latest techniques. Enjoy programming professionally with your own languages, and know what is happening with the others. Only PC Developer has the breadth and depth to help you write innovative, powerful and bullet proof applications. Subscribe to PC Developer magazine now - its the only way you can get it!

A FREE sample issue is now available. To get your copy, call 1-800-223-4064 (US) or fax 717-435-0375.

Outside of the US, email 76711,1243 with your address details for your free sample issue.

TO SUBSCRIBE NOW **BY MAIL OR FAX:**

- 1) Choose Print Commands from the File menu (Alt, F, P)
- 2) In the Print Commands box, choose the RoboPrint option
- 3) In the Select Help Topics box, choose the Selected Topics option
- 4) Choose the Range button
- 5) Select the Help Topic "SUBSCRIBE" and choose OK
- 6) Choose the OK button on the Print Help Topics dialog box
- 7) Mail or fax as per instructions on the form.

TO SUBSCRIBE NOW **BY EMAIL**

- 1) Select the "SUBSCRIBE" topic
- 1) Choose Copy from the Edit menu (Alt, E, C)
- 2) Paste the text into WinCIM or other email editor and send to CIS account 76711,1243.

SUBSCRIPTION FORM - PC DEVELOPER HELP FILE

Our Compuserve and physical contact details can be found at the bottom of this document.

Yes! I would love to subscribe to the following offer!

	US\$
Full Year Subscription (standard)	\$69
Full Year Subscription (pro - with disks)	\$139
Volume One (All previous issues + disks)	\$99

Name				
Company		Position		
Address		City		
State	Suburb		Zip	
Tel	Fax			
Email Address:				

Amount _____
Payment Type: _____
Money order / cheque enclosed
Govt Purchase Order
Credit Card
Card Type _____

Card Number _____

Signature _____

Expiry _____

Postal Address:

**PC Developer Magazine
RR4
Box 1001
Montoursville PA 17754-9433**

Tel 1(800)223 4064

Fax (717)435 0375

EMAIL CompuServe 76711,1243

[RETURN TO MAIN SCREEN](#)

The Articles

In depth technical articles that include source code fragments. Easy to read, well laid out and topical. Each article covers an interesting technique or programming issue. Here they are so far:

PC Developer Magazine List of Articles in Volume 1

Use these as a starting point of ideas for your own article. Just because you see an article listed, that does not mean you cant write about this area - but you must at least have a look at the previous article, refer to it, enhance it or add to it.

Generic

Using ODBC * Compiling Windows Help * ISYS DLL Opens The Document Door * Understanding Interface Design * Paradigm shift marks the end of our world * Automatic Application testing * Database Design and Normalization * Object Oriented Analysis * Windows Tip: DLLs Explained * Getting help from DRG * Acer Software Competition * Power to the People * Editorial - Power to the People? * OS/2 Developer Resources * Editorial - Borland bites back! * CompuServe Drops Prices, Expands Services * The Merits of Function Point Analysis * Consultants: Is Big Brother Taking Over? * Editorial: The times, they are a' changin'! * Frameworks & Infrastructures * Editorial: Programming Environments - the ultimate application * There's gold in them thar' virtual hills! * Legal issues in software development * Windows event handling: The basic principles * When OOP is not a mistake * Learning with PC Logo for Windows * Leveraging Off the Sybiz Accounting Engine (DDE) * Object Databases - Why?

VB - Easy to understand

Editorial - Future prospects for VB * Review: Visual Basic How-To * Review: The Visual Guide to Visual Basic * Book review: Advanced Visual Basic * Review: Visual Basic SQL Server Primer * MS Electronic Forms Designer * Break the 64k barrier with VB * The future of Visual Basic * A Sneak preview of VB95 * Development Standards for Visual Basic *

VB - Getting into it

Creating VBX-less 3D effects * Custom Interfaces for Standard Apps * Custom Interfaces for Standard Apps II * Using system and other sounds in VB * Pop Up Tool Tips MkII * Use of Windows Resources in VB * About Boxes in Access and Visual Basic * Optimising VB Code * VB Development Standards 2 * Public Access applications * VB 3.0 & Access Database Tips 1 * Stabilising Comm's Port Operations * How to use True Grid * Scroll Bars in Text Boxes * 3D Common Dialogs in VB Professional * Using objects in VB * Access 2.0 VB 3.0 Compatibility Layer * Finer Points with VB DoEvents * Visual Basic Animation * VB Print Wrap * How to use Network DDE * Common Dialogs for network applications * Out of Stack Space - Modal Forms * Linked List Data Structures in VB

VB - To the max!

Bui;ld your own image processor * Floating Popup Menus in Visual Basic * Setting ListIndex Properties * Easy Multi-Font Printing in VB * File Drag/Drop with Spyworks * Listboxes with Large Arrays in Visual Basic * A CAD Program in Visual Basic * The Order of Things (Fast Sorting for the Common Man) * Creating "Tiny" Title Bars in VB3 * GetPrivateProfileString() Tricks in VB * PopUp Tool Tips * Do-It-Yourself Compression * Providing enhanced functions in VB Text Boxes * Using VB to maintain memory pointers * Vector Drawing in VB * Calling NetWare API functions from VB * Working with TIFFs in VB * Visual Basic DOS Interrupts

Access - Easy to understand

Development Standards for Access * Access 2.0 is for Developers * Integrating Access 2.0 with Other Apps * MAPI Interface For Access * Code and Data MDB files * Home grown navigation controls in Access * Six Access Basic Optimisations * Optimizing Access

Access - Getting into it

A Quick Gateway Server * VB 3.0 and Access database tips * VB 3.0 and Access database tips II * ReQuery Access 2.0 Combo Boxes Automatically * Comparing Code - Access & dBASE * About Boxes in Access and Visual Basic * API Dialing from Access and Word * Splitting Access MDB files * Access, the Windows API and other DLLs * Error Handling in Access * Select Multiple Rows in Access Lists * Access Distribution Kit * List Management - merging data (part 1) * Upgrading Distributed Access 2.0 Applications * Turning Off Access Toolbars Using API * List Management in Access: Tune-able Deduplication - Part 2 * Documenting and Transferring Import/Export Specifications * VB 3.0 & Access Database Tips 1

Access - Power Programming

Access Wizards * Upgrading Access 1.1 Applications to Access 2.0 * Build Your Own Hypertext Databases * Access Speed Tips * Modifying Printer Settings within Access Apps * Providing Query-by-Form Power in Access * Inside Microsoft Access Wizards

Delphi/Pascal (Expanded in Volume II)

FirstLooks: Phase 3 * Getting More From Borland Pascal 7.0 * File Dialog Box: A TPW DLL * Turbo Pascal, Communications & FOSSILs * SIMCOM - A simple Turbo Pascal Communications Program. Part 1 * Embedding Data in a TPW .EXE * Embedding Data in a TPW .EXE * Embedding Text with Typed Array Constants and RC Data * Dialogs, Bitmaps & 3d Effects * Word Basic, Pascal DLLs & Word for Windows * Delphi on trial * Embedding Delphi forms in DLL's for use with VB * Borland Delphi: I'm impressed! * A Sneak Preview of Delphi 95

C++

ToolHelp Module Functions * Property-style Interfaces for C++ * Subclassing Windows for DOS Interface Compatibility * Product Review: Visual Classworks * FirstLooks: Phase 3 * A Class Library for SQL Server * Extending Common Dialogs * Animate your Program Icons * Owner-Draw Listboxes * Self Protecting Software * Clean Up Memory Using Destructors * Should we port MS DOS Programs to Windows? * Object Oriented Database: Evolution Or Revolution? * Profile File Class * Memory Allocation for MS DOS * First Looks: OEW for C++ * Posting Information to the Menu Bar * Virtual memory management in C++ * Intercept & Playback events at Device Driver Level * Binary Tree in C++ * First Looks: C++Creator version 1.8 * Hooks and Alternate Shells * C++ Compilers - An Independent Comparison * OOP - Encapsulation * OOVMM2 * OOVMM3 * Absolute Disk Sector I/O in C * DBLIB Access to SQL Server in C * OOVMM3 * Firstlooks: Watcom C++ * C: Does it have style? * Instant Keyboard Accelerators for Borland C++ * Book Review: Using Visual C++ * Review: Integra Visual Database for C++ * Object Oriented - Part 2: Polymorphism * Buffered files in Windows * Break into the world of encryption *

Office Integration

API Dialing from Access and Word * Word Basic, Pascal DLLs & Word for Windows * WordBasic and a GDI MessageBox * Pivot Tables in VBA *

Capturing and Storing Data with VBA in Excel

FoxPro

Centering Text in FoxPro * FoxPro's Foundation Read * Using Windows DLLs From Foxpro * Porting DOS Foxpro to Windows * Foxpro - A developers overview * A FoxPro Code Pre-Processor * My FoxPro Toolbox * FoxPro Pre-Processor II * Adhoc Report Writing with Foxfire! * Rebuilding the Power Tools * Distributable FoxPro spelling * Transaction Processing and Rollback in FoxPro * Leveraging Off the Sybiz Accounting Engine * News & New Products

Paradox

Joining Fields in Paradox * Paradox Reports * Using SendKeys with Paradox * Paradox for Windows - Methods and Procedures * Paradox for Windows - What's Happening? * Paradox: Handling Errors with Elegance * Graphing Dynamic Crosstabs * Floating Help Box DLL * PowerButtons for Paradox for Windows

Superbase

* Superbase V2 Tips and Tricks * Chaining Superbase Programs * Duplicate Superbase File Entries * Optimising Superbase Query Filters * Data Resource Manual for Superbase * SPC Dumps Superbase * Event-driven Programming under Superbase * Configuration Management in Superbase * Where is Superbase? * Execution in Superbase

Other

Disabling Print Screen * dBASE for Windows - The facts, not the hype * First Looks: OpenInsight * OS/2 and REXX * Writing Efficient Smalltalk Code * Writing Efficient Smalltalk Code * Data handling with Smalltalk Collections * Stripping and Packing Smalltalk Images: Part 1 * Stripping and Packaging Smalltalk Images: Part II * ToolBook Wizardry

New Sections for Volume II

CA Visual Objects

dBASE Windows

ToolBook

Gupta

Powerbuilder

Email all questions and submissions to 76711,1243

Base word rate is 14c including source.

Please include mailing address with all submissions for cheque payments.

SUBSCRIBE

CANCEL

Contributing Articles

Turn your spare time into dollars, and further your reputation and career at the same time. PC Developer is looking for quality technical articles. Are you good at expressing yourself? Have you been involved in some interesting projects? Can you write about a complex idea? Can you bank cheques?

If you are tempted...

You can send us a brief synopsis of the articles that you would like to write - about 50 words summarising the content of the article. Send us 4 or 5 alternatives to choose from - we may want them all! Send these summaries by email to 76711,1243. Make sure you make it easy for us to contact you, and include physical address details for cheques. Normal rates start at around \$200 per article - so get writing.

The best idea is too write about the interesting bits of a project you recently worked on - leaving out the names of the innocent etc.

As they say - publish or perish!

SUBSCRIBE

CANCEL

The Resource Diskettes

1.44 Mb floppies that are crammed full of executables, source code and data files that support and illustrate the printed articles. In addition, resource files, programmer' shareware and graphics for use in your applications are included for each PC Developer magazine issue.

And of course, you get the digital article database application (PCINDEX.EXE) to help you find exactly what you need, when you need it... **YESTERDAY!**

SUBSCRIBE

CANCEL

The Authors

In March, 1993, Microsoft introduced an award to recognize the frequent and valued contributors to product forums on CompuServe. The first annual Microsoft Support "Most Valuable Professional Award" (MVP), was presented to 34 of the most active members in recognition of the technical knowledge and experience they share voluntarily on the forums. Everyone benefits when experts share their knowledge and experience, and Microsoft wanted to recognize the MVP's for their past and future contributions to the lively technical dialogues. [Following are some of the MVP award winners that write for "PC Developer" magazine.](#)

Microsoft VISUAL BASIC

J.D. Evans
Gregg Irwin

Microsoft ACCESS

Ken Getz

Microsoft FOXPRO

Jim Booth

Other Respected PC Developer Authors

MS Access

Ashley Hubbard
Dr Rick Dobson
Michael Stringer
Stephen Moignard

Visual Basic

Jim Karabatsos
Eric Wilson
Evan Hopkins

C/C++

David Wediger
Philip Jones
Michael Zolotarev
Xiang Xiao
Craig Smith
Graeme Costin

FoxPro

Goran Zidar
Daryl Caithness
Dan Weinstein

Paradox

Brad Devison

Pascal

Roy McDonald
Ron Lyth

OOP

David Whittakers
Bruce Coleman
David Wediger

Noel Craske

Toolbook
Andy Bulka

[RETURN TO MAIN SCREEN](#)
[SUBSCRIBE](#)

PC Developer - What is it?

PC Developer magazine is easy to read and understand, yet tackles tough language specific and generic issues.

In depth articles look at maximizing your performance with Access, Visual Basic, C/C++, Pascal/Delphi, Paradox, FoxPro, CA-VO, SQL Server and more.

Indexed Resource diskettes let you play with source code and sample executables.

PC Developer now has 70+ pages packed with information to keep you up with the best. Sharpen your skills and make your code more efficient by learning the latest techniques. Enjoy programming professionally with your own languages, and know what is happening with the others. Only PC Developer has the breadth and depth to help you write innovative, powerful and bullet proof applications.

Subscribe to PC Developer magazine now - [its the only way you can get it!](#)

SPECIAL OFFERS

Every month we offer new premium offers, including free commercial software, books and product discounts.

These offers are language focused. Right now we have Delphi, C++, Pascal, VB, CA-VO and Access bundles - each with about \$300 of value to include with your subscription.

SO GET ON THE PHONE TO WYNNE YODER NOW!

ASK HIM WHAT THE BEST CURRENT DEAL IS AND HAVE YOUR CARD READY!

***** **1-800-223-4064** *****

[Full Year Subscription](#)

[RETURN TO MAIN SCREEN](#)
[SUBSCRIBE](#)

Contents and Using the Index Software

Using the PC Developer Article Database Program to find PC Developer articles.

This program will assist you to find specific articles covering any topic published in PC Developer without your having to look through the magazines or index files manually.

It is simple to use:

- 1) **Select a radio button to sort articles by Author, Language, Title or Keywords.**
- 2) **Double click in the combo box or type in the first few letters of the search criteria.**
- 3) **Double click in the bottom table listbox to select the exact article you are looking for - its details will appear in the notes window on the right hand side of the screen.**

In addition, you can print the search results or copy them to the Notepad Windows application for use in other programs.

[RETURN TO MAIN SCREEN](#)
[SUBSCRIBE](#)

its the only way you can get it!

That's right - **PC Developer** magazine is not available on the news stand - you must call us to get it! Use the following number in the US -

1(800) 223 4064 or Facsimile (717) 435 0375

or mail to

**PC Developer magazine
RR4
Box 1001
Montoursville PA 17754-9433**

(Print the "SUBSCRIBE" Topic from the File menu...or copy it to the clipboard for Email)

SUBSCRIBE

CANCEL

Full Year Subscription

EVERY MONTH WE HAVE SPECIAL SUBSCRIPTION OFFERS RUNNING. CURRENTLY THERE ARE SPECIAL OFFERS OPEN TO DELPHI, ACCESS, VB, FOXPRO, C++, PASCAL and CA-Visual Objects PROGRAMMERS!

CALL WYNNE YODER NOW ON 1-800-223-4046 FOR DETAILS OF THE
SPECIAL SUBSCRIPTION OFFER FOR YOUR TOOL.
FAX - 717-535-0375

ASK FOR YOUR FREE SAMPLE ISSUE!

PROFESSIONAL SUBSCRIPTION OFFER

PC Developer Binder with 12 Diskette Sleeves
PC Developer Mouse mat
PC Developer Magazine Back issues (11)
PC Developer Resource Diskettes Back issues (11)
PC Developer 1995-96 Subscription Issues (10)
PC Developer 1995-96 Subscription Resource Diskettes (10)

A total package normally costing \$250.00-for a limited time only the kit is yours for **\$199.00!**

STANDARD SUBSCRIPTION OFFER

PC Developer 1995-96 Subscription Issues (10) - Total price **\$69**

SUBSCRIBE
CANCEL

(Access) Error Handling in Microsoft Access

****SAMPLE ONLY****

If you never make mistakes and you have perfect users, read no further. You won't be interested in handling imperfection. However despite my best efforts and a rigorous QA process, I haven't yet produced an error-free application. Unluckily too, my users make mistakes and their computer environments are unreliable. And I'll wager that all Access developers have these problems (95% admit it and the other 5% are liars). All this leads to run-time errors, which when not handled properly can stop a good application dead. Effective error trapping though, can improve your quality assurance work and the resultant application.

Error handling is one of the benefits of using Access Basic instead of macros. When macros fail because of run-time errors, they fail totally. You have no control. This looks unprofessional and you cannot tidy up after the error, leaving the application in a dangerous state. If there is any possibility of a run-time error, you should use Access Basic for procedural functionality. You may find that the function just a series of DoCmd statements calling macro actions, but the error trapping makes all the difference.

Access, like Visual Basic, requires that you design and code individual error trapping for each function. (Though this article will use the term 'function, all the information also applies to procedures.) On first impressions this seems cumbersome, but in many situations the context of the function determines how a specific run-time error should be handled. You will probably need to handle the same error differently in two distinct functions and this will need different code.

Tips and Traps for New Players

Having said that an advantage of Access Basic is error trapping, the first tip is avoid writing unnecessary error trapping code. Some developers have a policy of never using macros, which means their applications have many short functions executing macro actions. There is normally no chance of users experiencing a run-time error in these simple functions if proper testing has occurred. Thus writing error trapping code just consumes time, negating one of Access major benefits {SYMBOL 190 \f "Symbol"} high developer productivity. You can easily add error trapping code later if testing reveals the possibility of errors.

The lack of error trapping is not a good reason never to use macros. Many Access developers, myself included, think macros are a major benefit of Access. Used in the correct context they provide high development productivity. With macros you produce more application in less time. The second tip is use a naming convention for error trap and function exit labels. As the number of functions in your application grows, it is possible to create duplicate error trapping labels in the same module. A tight naming convention prevents this. We use the Leszynski/Reddick convention of appending the word 'Error' or 'Exit' after the function name to create the label, as shown following.

```
Function fnName ()
    On Error GoTo fnNameError
    Dim strMsg as String
    <function code here>
fnNameExit:
    <tidy up code e.g. closing queries and tables>
    Exit Function

fnNameError:
    <error handling here>
    strMsg = "Error in 'fnName':" & Chr(13) & Chr(10) & Err & " " & Error
    MsgBox strMsg, 16, "Application Name"
    Resume fnNameExit
End Function
```

Tip number three:

Use a global variable to record whether the application is operating in a development environment or at the users site. This allows one AutoExec macro for all situations and helps control error handling. For example, when developing an application we don't want the menu form to appear and the database container to disappear. The first statement in the AutoExec macro sets the global variable gfDebugMode. The condition column for actions that you

don't want to occur in development call a function `debugMode()`, which returns true or false.

Tip number four:

On CompuServe there is a text file (ERLIST.TXT) that lists all possible run-time error codes in Access. Download it for easy reference when an error occurs. There is a copy on the Resource Disk.

The last tip is use your own message box for displaying unhandled run-time errors. The default Access message box for errors in Access Basic doesn't display the error number or the function in which the error occurs. You need the error number for coding handlers and the function name gives you a starting point for defect location. Without error trapping in a function, Access displays a message and then opens the module at the line at which the error occurred. However, if you have secured your application so users cannot view the modules, they also can't see where the error occurs (and you won't want them to), thus causing another run-time error. The example function above shows a useful message box for errors.

Error Log

The first step in more sophisticated error handling is an error log table. We use the error log when testing an application and for maintaining it after users have it on their systems. The error log records each run-time error that occurs, providing you with an invaluable history of what went wrong. An error log table (zstblErrorLog) requires at least the following fields:

DateTime		Date/Time
User		Text
FunctionName	Text	
ErrorNumber		Integer
ErrorMessage	Text	

Though you can retrieve an error's message using the error number, we store the generated message. This is because at the moment of error, Access inserts the name of the object causing the error in many messages. If you don't store the generated message, you lose the name of the object. When displaying the message later using `Error` (`ErrorNumber`), Access displays a vertical bar in place of the object name. To make use of the error log, include the following code as part of a function's error handler:

```
strMsg = "Error in 'fnName': " & Chr(13) & Chr(10) & Error & ". Log?"
If MsgBox(strMsg, 308, gAPP_TITLE) = 6 Then
    logError "fnName", Err, Error, 0, ""
End If
```

We give users the option to log errors when they occur, though I am not sure that this is a good thing. We also use the error log to note the key identifiers of records with errors. This is what the last two parameters for the `logError` procedure handle. Manipulating record sets with `Do` loops or special validation queries can reveal individual records with errors. With the key details stored in the error log, users can then examine the individual records to repair problems. Our applications include an error log report so that users can view or print the error log.

Error Handling Function

In larger applications, you will find that particular run-time errors can occur in many functions. Errors occur when users try to access objects for which they don't have security permissions. File handling also generates many errors. In these situations you will want to use the same error handling code in many functions. To save time, you can store this code in an error handling function and call it from each function's error trapping code.

An error handling function provides a consistent method of dealing with common run-time errors. This makes for consistent messages, higher quality, easy maintenance and quick internationalisation of an application (changing the application user language e.g. from English to French). The benefit alone in consistent messages is significant. Users quickly notice different messages for the same error or task, and generally think less of an application for it.

An error handling function 'controls' the error handling in the calling function, the function with the error, by returning a value. The calling function take specific action in response to the returned value, normally by using a `Select Case` structure as shown in the function below.

```

Function fnName2 ()
    On Error GoTo fnName2Error
    <code goes here>
fnName2Exit:
    Exit Function

fnName2Error:
    <error handling code specific to this function>
    ' Analyse any other error and decide how to handle it
    Select Case handleError("fnName2", Err, Error, 0, "")
        Case 0
            Resume
        Case 1
            Resume Next
        Case 2
            Resume fnName2Exit
        Case Else
            MsgBox "Critical error! Quitting immediately!", 16
            DoCmd Quit A_EXIT
    End Select
End Function

```

Return values depend on how you code the error handling function, which depends on how you want the application to handle the passed run-time errors. You normally use a Select Case structure testing the passed error number to determine the appropriate return value. Possible return values from handleError() are 0, 1, 2 and 3. These codes recommend that the calling function take one of the following actions respectively: Resume, Resume Next, Resume function exit or Quit. Also included in the error handling function is code to display an error message and to call the logError procedure.

Of course the calling function can ignore the return value and take specific action in response to any error. Before you call the error handling function, you can insert error handling code specific to the calling function. For example, in most situations calling a non-existent function would be an error justifying exiting the calling function. But in menus, listing non-existing functions allows you to display future functionality to the user. Coding specific error trapping for the situation on the menu selection function displays the appropriate message when the user tries to access that functionality.

The usefulness of an error handling function depends on the complexity of the application and the number of times a common task occurs in different code. The handling of run-time errors often depends on the context of the function in which they occur. You may want to handle the same error differently in two distinct functions. This reduces the benefits from an error handling function.

Application-Specific Errors

With a little more planning, you can also use an error handling function to handle application-specific errors in a consistent manner. Using the Error statement and your own error numbers, you can use Access error trapping to handle application errors. By placing code to handle application errors in the error handling function, you again get consistent messages and easy internationalisation. The example below shows how this works for referential integrity:

```

Function frmName_OnDelete ()
    On Error GoTo frmName_OnDeleteError
    'This line checks for a record in tblChild and generates an error
    'if there is one.
    If Not IsNull(DLookup("ParentKey", "tblChild", "ParentKey = {SYMBOL 222 \f
    "Symbol"} Form.Key")) Then Error 32760

frmName_OnDeleteExit:
    Exit Function

```



```

frmName_OnDeleteError:
  Select Case handleError("frmName_OnDelete", Err, "", 0, "")
    Case 2
      DoCmd CancelEvent
      Resume frmName_OnDeleteExit
    Case Else
      MsgBox "Critical error! Quitting immediately!", 16
      DoCmd Quit A_EXIT
  End Select
End Function

```

Microsoft recommends that you count down from 32,767 when assigning error numbers to application-specific errors. In the error handling function, you add code to the Select Case structure to deal with these error numbers. This includes assigning an error message to the error and displaying the message to users. You can also log application-specific errors if you want to. As you can see, improving the way in which your application handles run-time errors improves the quality of your application and makes it more robust. It also reduces the time testing and repairing defects. Most importantly, improving error handling reduces support requirements, which builds better relationships with your users.

Issue	PCD/5
Title	Error Handling in Access
Article ID	88
Complexity	6
Keywords	error handling, run-time errors, error log, Access Basic
Language:	ACCESS
Operating System	Win 3.x
Author	Michael Slinger
Network ID	110026,2466

[**RETURN TO MAIN SCREEN**](#)
[**SUBSCRIBE**](#)

Listboxes with Large Arrays in Visual Basic

**** SAMPLE ONLY ****

Problems arise when trying to use a standard listbox with large arrays, particularly if data has to be read from large database tables. This is because memory has to be addressed in 64K blocks, thus controls such as the standard listbox are limited to 64K of data, an unfortunate leftover from Dos land, that will not be fixed till a true 32 Bit operating System comes along, ala Chicago and NT.

Lets suppose you have a database table with two thousand records and want to load that data into a standard listbox. Assume we have a fifty character field length, then you can load about $(64*1024)/50 \sim 1310$ records into the listbox. Even if you could load two thousand records, the time taken to load even such a large amount of data into the standard listbox would be quite substantial, and would depend on the performance of the computer and the size of the database table. So reading portions/pages of the data into a control as required is the only practical way of using a standard listbox for large amounts of data, to overcome our 64K restriction.

Graphical design:

The efficiency of the listbox depends on the actual portion of the data which is read in. There are two different ways of defining portion :

1. The page of data is constant all the time.
2. The page of data depends on the actual position within the listbox.

The first case is the simplest to implement, however it doesn't work efficiently enough for boundary conditions. Suppose you are about to leave the data portion border within the listbox, In which case the listbox will have to read the next page in its entirety, instead of reading just one record. Once loaded you are still on the border but at the other side of the page. If you go backward to the previous record, you will need to read the previous page of the data back in again ! This functionality can be quite annoying for the user of the application .

The second case is quite powerful, however it requires a far more complicated implementation. Because we are dealing with three objects of data (Table data, Page data and Visible data) we have to support control for the combination of these objects' boundaries.

Now the standard listbox does not have any events to control the listbox scrollbar. This makes it hard to control boundary position when we try to scroll the listbox with the mouse using the listbox scrollbar (ListIndex stays the same).

So as a work around, add an additional scrollbar to the listbox. This will overlap the existing listbox scrollbar. Now we have a Scroll_Change Event which allows us to follow all movement within the page.

To implement this we will first need to define a type which will be used to store all of the Virtual listbox Parameters.

```
Type RecordWnd
Type RecordWnd
    DynaLength As Integer 'Length of Dynaset or table
    ListLength As Integer 'Max Length of listbox (= ListCount)
    ListBegin As Integer 'Beginning of list Area in the Dynaset (table,
Dynaset, snapshot)
    ListWinSize As Integer 'size of visible part of listbox
    ListTop As Integer 'Top of visible part of list regarding to
recordset
    ScrollPoint As Integer 'Current Value of Scroll (point of current record)
    Separator(10) As String 'Separator in the formatted string
    Item(10) As String 'Name of field in the dynaset for formatted
string
    Frmt(10) As String 'Format of record
End Type
```

Parameter definitions :

DynaLength - RecordCount of database table
 ListLength - actually it is the size of data portion
 ListBegin - is the number of record in Table which is the first record in the portion.
 ListWinSize - size of visible part of the listbox
 ListTop - Number of record which is at the top of the listbox
 ScrollPoint - Current Value of Scroll (equals to the current record number)

The RecordWnd Type has three additional variables to define the appearance of the listbox , and allow multiple fields to be displayed. They define the list of field names (Item) , the fields format (Frmt), and finally the fields separators (Separator). In this example the arrays have 10 elements, however they can be declared dynamically using the REDIM Preserve Statement.

The MS Access engine does not give us the current record number, however it does provide bookmarks. So before loading the data into the listbox , an array of table bookmarks is created., and the number of the current array element reflects the number of the current record in the table. By using the bookmark array we can speed the data loading process up, otherwise we have to use the FindFirst, FindNext methods with Dynasets can get quite slow. (Even theSeek method with tables is slower than bookmarks in this situation)

Example of loading the bookmark array :

```

'
' Set Record Bookmarks Array
'
lst.MoveFirst
ReDim RecPoint(lst.RecordCount)
For i% = 0 To lst.RecordCount - 1
    RecPoint(i%) = lst.Bookmark
    lst.MoveNext
Next i%
  
```

The following piece of code Initialises the Listbox parameters,

```

'
' Set properties for Virtual ListBox
'
rwTest.DynaLength = lst.RecordCount
rwTest.ListLength = 50           'ListBox Portion
rwTest.ListBegin = 1
rwTest.ListWinSize = 21         'Listbox Visible Portion
rwTest.ScrollPoint = 1

'
' Layouts for the Listbox
'
rwTest.Separator(1) = ""
rwTest.Item(1) = "ID"
rwTest.Frmt(1) = "####"
rwTest.Separator(2) = "      "
rwTest.Item(2) = "List record"
rwTest.Frmt(2) = "#####"
'
' Set VScroll
'
VScroll1.Min = 1
VScroll1.Max = rwTest.DynaLength
VScroll1.LargeChange = rwTest.ListWinSize - 1
  
```

Now the following code actually loads the listbox,

```
ListScroll List1, VScroll1, lst, rwTest, 2, RecPoint(),
```

where,

Parameter	description
List1	Listbox control
Vscroll1	Scrollbar control
lst	Table object
rwTest	RecordWnd variable
2	Mode - initialisation

Intelligent loading of listbox data:

We use the Scroll_Change Event to control the ListBox. As soon as the scroll value reflects the actual current record number in the table, we load that portion of the table data into the listbox using the procedure ListScroll().

The data portion depends on the currently selected items position within the listbox, ie the Data -Displacement between the old Scroll_Value and the new Scroll_Value.

The new portion is only unloaded when the Scrollbar's current portion is out of bounds. Displacement is calculated using ListBegin, ListLength, ScrollPoint, and ScrollValue and takes into account the DynaLength value. In any case, the portion cannot be more than the displacement or ListLength. For example, if you want to move one record further and have gone out of boundary, only one record will be uploaded. The table below shows the portion dependence.

Displacement	ListLength	Records will be uploaded (Max amount)
1	50	1
20	50	20
100	50	50
100	100	100

The speed of the Virtual Listbox depends on the size of the data portion. The maximum data portion depends on the ListLength parameter. It is a flexible parameter and can be changed depending on user actions. The suggested Minimum can be determined from the following calculation, $Listbox.Height / text.Height$. Otherwise you will have a half filled listbox. The suggested maximum would be not to have any more than 500 records (can be less if you have long strings).

Issue	PCD/5
Title	Listboxes with Large Arrays in Visual Basic
Article ID	93
Complexity Level	7
Keywords	LISTBOX, ARRAY, 64KB, DISTRIBUTION TOOLKIT
Language	VISUAL BASIC
Operating System	Win 3.x
Author	Terentjev, Eugene and Hopkins, Evan
Network ID	76711,1243

[**SUBSCRIBE**](#)

[**RETURN TO MAIN SCREEN**](#)

Hooks and Alternate Shells in C/C++

SAMPLE ONLYr

I hate Program Manager.

OK, it does the job, but its way too big, and screen space is always at a premium, even running at 800x600 (no I havent taken delivery of my 20 dream monitor just yet). So I did what all programmers do when they dont like a particular program - I wrote one for myself. This month I will describe how I developed my alternate shell, called WinMen.

The most interesting feature of its display is that it doesnt have one, so it takes up very little room on the screen. When I click with the right mouse button on the desktop, it pops up at the current mouse position, displaying my set of programs as a hierarchical menu, from which I can choose something to run. One of the options even pops up a little dialog where I can type the name of a program to execute! Im already ahead of ProgMan on two counts: far less screen space, and nested program groups.

The menus are defined in a single file in the users Windows directory, called WINMEN.DAT. The level of a menu item is defined by its indentation (using the TAB character), so the file may look something like this:

```
&Apps
    &Utilities
        &Control Panel,control
        &Notepad,notepad
    &Word,c:\winword\winword
&Devel
    &Visual C++,c:\msvc\bin\msvc
    &AppStudio,c:\msvc\bin\apstudio
```

A line which has a single field represents a popup menu. If it has more than one field, it is a menu item. You can specify the program to run, and using an optional third field, the directory to run it from. It also supports a simple form of macro substitution, which prompts for a parameters to be substituted for the macro. For example, the command

```
&Word,c:\winword\winword #(Document Name)
```

will pop up a very small dialog box with the prompt Document Name, and substitute whatever you type as the command-line arguments to WinWord.

Implementation.

There are two key API sets I have used here. The first is TrackPopupMenu, which pops up a menu anywhere, anytime. To use it, you must first construct a dynamic popup menu (you could also a submenu of a an existing menu loaded from a resource). The starting point is CreatePopupMenu, which returns a handle to an empty popup menu. You must then call InsertMenu or AppendMenu to attach things to it. Depending on the parameters to either of those functions, what you attach can be either be a menu item, or another popup menu, so you can build nested menus quite easily.

When the program starts up, it scans the WINMEN.DAT file, and builds the popup menu. When it is asked to pop up its menu, it calls GetCursorPos to retrieve the mouse coordinates. These are passed to TrackPopupMenu to say where the menu should pop up.

Another parameter to TrackPopupMenu is the ubiquitous hWnd, which determines which window gets the resultant WM_COMMAND message that arises from any menu click. But where is my application window? The answer is that its a hidden window. Any application that wants to stick around for more than a very short time must have a window and a message loop, but if you dont want anyone to see it, you must make it hidden, simply by not calling ShowWindow, or by not including the WS_VISIBLE style in the CreateWindow call. Other than that, the main message loop is identical to any other Windows application. But of course, the big question is, if the window is hidden, how does it get any user input? How do we tell it to pop up a menu?

Hooks.

The answer is by using a hook. Hooks are a fantastically powerful feature of Windows, allowing any application a chance to peek at any system event, window message, or keyboard event. They can also be used to implement macro

recorders (like Windows Recorder, Microsoft Test etc). As I just read the other day, they can even be used to compromise the security of a well-known, allegedly C2 compliant operating system (I won't mention it by name). You set up a hook using the SetWindowsHookEx API. This replaces the pre-Windows 3.1 SetWindowsHook call, which still works, but should no longer be used. You must specify the type of hook, a callback function, and a couple of other parameters, as we will see.

Here are just some of the hook types you can pass.

Hook Type	Meaning
WH_CALLWNDPROC	window-procedure filter
WH_CBT	computer-based training filter
WH_GETMESSAGE	message filter
WH_JOURNALRECORD	journaling record filter
WH_JOURNALPLAYBACK	journaling playback filter
WH_KEYBOARD	keyboard filter
WH_MOUSE	mouse-message filter
WH_MSGFILTER	message filter

Each of these functions has an associated callback function which details what the parameters are, and what can be done with the event. A hook can either be task-wide, i.e. applying to the current task, or system-wide, applying to all tasks. If it is only a task hook, it can reside in an EXE file, otherwise, it must be in a DLL so every task has access to it. The documentation states that certain hook types must be in DLLs, but this is incorrect. Only system-wide hooks must be in DLLs.

Lets say you wanted to trap all WM_PAINT messages in the system, and cause every third one to magically disappear. This will have a devastating effect on the system, and is great for really annoying people(no, I don't know how to set up hooks across a network). The first thing you must do is establish the hook:

```
static HHOOK hMsgHook;
extern HINSTANCE hInst;
extern int CALLBACK GetMsgProc(int, WPARAM, LPARAM);
...
hMsgHook = SetWindowsHookEx(WH_GETMESSAGE, GetMsgProc, hInst, NULL);
...
```

The last parameter of NULL indicates that the hook is not bound to a particular task, but rather is a system-wide hook. You must then write the actual hook function, which in this case is called before any window message in the system is processed. If the code parameter is less than zero, you must call CallNextHookEx, which implements the hook chain. Otherwise, you can do whatever you like with it. We have a counter that is incremented every time a WM_PAINT message is received by any window, and at every third one, it pokes WM_NULL (the null message) into the message field of the MSG structure, thus causing the message to effectively disappear.

```
LRESULT CALLBACK GetMsgProc(int code, WPARAM wParam, LPARAM lParam)
{
    if (code < 0)
        return CallNextHookEx(hMsgHook, code, wParam, lParam);

    MSG FAR *lpMsg = (MSG FAR *) lParam;
    static UINT count = 0;
    if (lpMsg->message == WM_PAINT && count++ % 3 == 0)
        lpMsg->message = WM_NULL;

    return 0;
}
```

When you're finished amusing yourself, you must unhook, as follows:

```
...
```

```
UnhookWindowsHookEx (hMsgHook);
...
```

A few words of warning when you're experimenting with hooks. As you can see, they can be very dangerous, and you will probably crash a few times. If your hook is in a DLL, it may not get unloaded (in a future article, I'll describe a utility for doing this). You may bring down Windows as well.

Trapping the mouse.

For my purposes, I wrote a small DLL called WHEN.DLL, which sets up a various types of hooks, and when they occur, posts a message to a particular window. The API of my DLL is very simple:

when.h

```
#define MH_MOVE      0x0001
#define MH_RBUTTON  0x0002
#define MH_LBUTTON  0x0004
#define MH_DBLCLK   0x0008
#define MH_DESKTOP  0x0010
#define KH_CTRL     0x0020
#define KH_ALT      0x0040
#define KH_SHIFT    0x0080
```

```
void CALLBACK stopWhen(int eventId);
int CALLBACK whenKey(WORD key, WORD event, WORD msg, HWND hWnd);
int CALLBACK when(WORD event, WORD msg, HWND hWnd);
```

You call the **when** function to set up an mouse event hook. The event parameter is formed by ORing together some of the MH_* symbols, so to set up a mouse hook for whenever the right mouse is double-clicked, you pass MH_RBUTTON|MH_DBLCLK. The **msg** parameter specifies what message is sent to the window (specified as **hWnd**) when the event occurs. The return value is a magic cookie which can be passed to **stopWhen** to terminate the hook.

The **whenKey** function does the same but for keyboard events. If you want to trap the Alt-8 sequence on a system-wide basis, pass 8 for the **key** parameter, and KH_ALT for the **event** parameter.

When follows is a skeleton of WinMens window proc, indicating how it uses the when API:

```
LRESULT CALLBACK WndProc(HWND hWnd, WORD message, WPARAM, LPARAM)
{
    switch (message) {

        case WM_CREATE:
            when(MH_RBUTTON|MH_DESKTOP, WM_USER, hWnd);
            break;

        case WM_USER:
            // go the mouse click
            GetCursorPos(...);
            TrackPopupMenu(...);
            break;

        ...
    }
    return 0;
}
```

And that's about all there is to it. To install WinMen as your replacement shell, you must alter the **shell=** entry in the **[boot]** section of **system.ini**. The files **winmen.exe** and **when.dll** must be in the Windows directory. A second word of warning: your replacement shell must be rock solid. If it crashes, you may have no way to run programs (unless

File Manager is running), you will have no way to exit Windows except with Ctrl-Alt-Del, and you may have to go back to a working shell in order to recompile it and test it further. You may want to develop on one machine, and test on another.

The main lesson from this exercise is that whenever you can, package up a set of functions into something that is reusable. I could easily have written a tiny DLL which just set up the hook that I wanted, and posted a message back to my application window, but that would have been useful only to that application. Instead, I have written a very generic wrapper around the Hook APIs, so that in future, I dont need to write yet another little DLL that does almost the same thing.

How to get it.

While I was thinking about this article, I decided this program is useful enough to be shareware, so I will post it in the WINSOL CompuServe section. For a small contribution, you will receive the latest version, with source code, and an update when I decide to make some enhancements. Some of the items on the cards includes

- Import of ProgMan .GRP files
- Ssupport for ProgMan DDE (for installation programs)
- Display of programs using icons as well as text
- A decent configuration dialog (with drag and drop support)

Needless to say, I will be keen to discuss the implementation of some of these features in upcoming articles. Sure, this program shell is no Norton Desktop for Windows, but Im a firm believer in Occams Razor: this application is small, simple, and does its job quite well.

What ever happened to...

When I started this series, I said Id have more to say about Actor in the future. Well, the future has arrived. The Actor development system has been acquired (from Symantec) by Genesis Development, an OOP specialist company in Philadelphia. As a Symantec distributor, it is not my place to say much about what Symantec did or did not do with the product, but Genesis are committed to enhancing the product. They are working on improved visual programming and group development facilities, integration with object analysis and design tools, support for networking and client/server computing, object storage and transaction management, support for OLE 2.0 and the Component Object Model, and conformance to Object Management Group (OMG) standards for enterprise-wide object computing. They clearly have some big plans for the product, and Ill keep you informed.

Issue	PCD/5
Title	If You Hate the File Manager - Hooks and Alternate Shells
Article ID	100
Complexity Level	8
Keywords	FILE MANAGER, RUN, WINDOWS HOOKS
Language	C/C++
Operating System	Win 3.x
Author	David Werdiger
Network ID	76702,2052

[SUBSCRIBE](#)

[RETURN TO MAIN SCREEN](#)

A FoxPro Code Pre-Processor

SAMPLE ONLY

One of the truly good things about the xBASE environment was the ease of programming that was possible; you make one variable whatever you like whenever you like. This was wonderful for those times when a quick program was needed to calculate one thing or another. I said 'was' because the day has come when large full scale applications are now being written in these xBASE languages, and that very freedom is becoming a time consuming liability for programmers. FoxPro has good functionality for scope limiting variables, as indeed do most xBASE type languages. PUBLIC, PRIVATE and REGIONAL commands provide the developer with a reasonable array of scopes for their variables to reside in. The down side to this is the often tedious task of maintaining these constraints. The temptation is thus to forget about these things and whack in a new variable wherever required. This is most often the case in maintenance programming, which makes the task of consolidating changes into new releases that much more difficult. How many of us have spent hours before the keyboard wondering why a given function behaves strangely, only to discover that one of the utility variables (ln_count, mcount... etc) was not declared as PRIVATE at the top of the function. Meaning that the value of this variable was being modified in the child as well as in the calling program, giving us these seemingly unexplainable errors. So what is the solution? Spend more time adhering to strict coding standards, excellent in theory but not very beneficial to productivity, or use a tool which automatically implements some of these coding standards.

As you read this article you may get the impression that we are blowing our own trumpet. Well, we are, but its for the greater good! There are many frustrating things about programming, not the least of which is debugging mistakes that you don't know exist. What we propose in this article is a program that is intended to make your life easier. One thing that it is important to keep in mind that this article is not intended to demonstrate the correct way to write programs, for that ethereal notion means different things to each and every one of us. What we are attempting to illustrate is a way which you, the programmer/developer can write applications which make coding your other applications easier. It must be kept clear that the ideas presented here are our ideas because this is the way we do things. They may or may not be suitable in your view. The idea is to get you thinking about using the flexibility of FoxPro to improve the way you develop your code.

In any programming language, with the possible exception of FORTRAN and Assembler, it is possible to write utilities which will allow you to automatically modify your source files before they are compiled. FoxPro, however, not only provides good commands for manipulating text variables and files, but the environment itself encourages the use of such tools. If it can provide a screen generator, a report generator, and a menu generator, then why not a code processor?

The use of a code pre-processor to enhance your productivity is not a new idea, in fact many editors offer just such a tool as part of their environment. What we are talking about here is an integrated pre-processor written in FoxPro to assist in writing FoxPro applications which are reliable and follow your own strict coding standards.

The pre-processor, in whatever form, is next to useless if it is inaccessible while writing your code. It is therefore important to make it available at call by either attaching it to the System Menu or specifying the pre-processor in the _BEAUTIFY System memory variable (this will of course prevent you from using the actual Beautify Application on your code). The latter is obviously the easiest of the two and can be implemented in your config file. The method for adding custom menu options to the System Menu can be useful if you wish to add more than one utility to your development environment. The easiest way to do this is by using the menu generator and creating a "Quick Menu". Then attach your own pads and replace the System menu with this generated menu by calling it in the _STARTUP section of the config file.

There is however a third option, that being to embed the pre-processor in a custom written FoxPro Integrated Development Environment. A discussion of just such a solution could be the focus of a future article.

One of the benefits of a pre-processor could be to remove the need for the programmer/developer to concern themselves with the tidiness of the application, allowing them to concentrate on improving the overall functionality of the system. The pre-processor could take control of the hack work of variable declaration and scope limiting, either automatically or with limited instruction from the user.

It could also write the code which validates the parameters and/or calls your audit routines, all the things that you would like to have in your code, but really could not be bothered keying in. If the capacity for using a variable wherever and whenever you like exists, then why not take advantage of this and let the pre-processor worry about correctly assigning scopes and variable declaration. In addition to this it would be possible to enhance the language

using the pre-processor. You could have the ability to write shorthand code which the pre-processor would expand into equivalent FoxPro code when encountered. One example which leaps quickly to mind is the C language's 'mcount++' command. This is far quicker to code than the FoxPro equivalent 'mcount = mcount + 1'. The user of the pre-processor could, therefore, have a whole library of such abbreviated commands or indeed functions which the pre-processor would insert into the final source program.

The way the pre-processor generates the final source program would be to read your original file line by line then outputting the resultant code along with any directives expanded and variables declared and verified. The following is a quick example of what a pre-processor could do:

Before the Pre-processor

```
FUNCTION counter
  PARAMETERS pn_start, pn_end

  *\ This function does nothing other than to
     illustrate some of the possible uses of
     a pre-processor, including expanding out
     comment start and end markers *\

  IF pn_start > pn_end
    ll_return = .f.
  ELSE
    ll_return = .t.
  ENDIF
  DO WHILE pn_start < pn_end
    pn_start++
  ENDDO
RETURN ll_return
```

After the pre processor

```
FUNCTION counter
  PARAMETERS pn_start, pn_end
  *{{ Inserted by pre-processor
PRIVATE ll_return, ll_ok
ll_return = .f.
ll_ok = .F.
DO CASE
  CASE TYPE("pn_start") != "N"
  CASE TYPE("pn_end") != "N"
  OTHERWISE
    ll_ok = .T.
ENDCASE
IF NOT ll_ok
  [<warning>] && User definable in setup of pre-processor
  RETURN
ENDIF
*}} End of insertion

* This function does nothing other than to
* illustrate some of the possible uses of
* a pre-processor, including expanding out
* comment start and end markers

IF pn_start > pn_end
  ll_return = .f.
```

```
ELSE
  ll_return = .t.
ENDIF
DO WHILE pn_start < pn_end
  pn_start = pn_start + 1
ENDDO
RETURN ll_return
```

There are a whole host of functions that you can get your pre-processor to perform, it just depends on your coding style, and what facilities you would like to automate or have the pre-processor simplify. The fact remains, in the end it should be your pre-processor, because it will be your code which it will act upon. As with the screen and menu generator it may start out as something useful and grow, with your enhancements or deletions into something perfect for you, because YOU are the one using it.

Issue	PCD/5
Title	FoxPro Code Pre-Processor
Article ID	103
Complexity Level	7
Keywords	PUBLIC, PRIVATE, REGIONAL, CODE PRE-PROCESSOR
Language	FOXPRO
Operating System	Win 3.x
Author	Philip Caithness and Goran Zidar
Network ID	100241,35

[SUBSCRIBE](#)

[RETURN TO MAIN SCREEN](#)

Free_Issue

A **FREE sample issue** is now available. To get your copy, call
1-800-223-4064 (US) or fax 717-435-0375.

Outside of the US, email 76711,1243 with your address details for your free sample issue.

PC Developer magazine is not available on the news stand - you must call us to get it! Use the above numbers in the US or mail to

PC Developer magazine
RR4
Box 1001
Montoursville PA 17754-9433

Advertising_rates

PC Developer Magazine US Advertising Rates

FOR MORE INFORMATION or a sample issue
Email Stephen Moignard on 76711,1243

Ad Type **1 Issue 2 Issues 3 Issues 6 Issues 10 Issues**

SPOT COLOUR - MAGENTA (where available) or MONOCHROME

Full Page	\$900	\$810	\$753	\$716	\$687
Back Cover	\$1,170	\$1,053	\$979	\$930	\$893
Inside Back Cover	\$900	\$810	\$753	\$716	\$687
Center Spread	\$1,350	\$1,215	\$1,130	\$1,073	\$1,031
3/4 Page Horiz	\$675	\$608	\$565	\$537	\$515
3/4 Page Vert	\$675	\$608	\$565	\$537	\$515
1/2 Page Horiz	\$450	\$405	\$377	\$358	\$344
1/2 Page Vert	\$450	\$405	\$377	\$358	\$344
1/3 Page Horiz	\$360	\$324	\$301	\$286	\$275
1/3 Page Vert	\$360	\$324	\$301	\$286	\$275
1/4 Page Horiz	\$270	\$243	\$226	\$215	\$206
1/4 Page Vert	\$270	\$243	\$226	\$215	\$206
Sponsor Panel x1	\$143	\$128	\$119	\$113	\$109
Sponsor Panel x2	\$203	\$183	\$170	\$161	\$155
Sponsor Panel x3	\$305	\$274	\$255	\$242	\$233
Sponsor Panel x4	\$406	\$366	\$340	\$323	\$310
Sponsor Panel x5	\$508	\$457	\$425	\$404	\$388
Sponsor Panel x6	\$609	\$548	\$510	\$484	\$465
Sponsor Panel x7	\$711	\$640	\$595	\$565	\$543
Sponsor Panel x8	\$812	\$731	\$680	\$646	\$620
Sponsor Panel x9	\$914	\$822	\$765	\$727	\$698

Art to be supplied as film positive, emulsion up

Bromides can be supplied - additional charge - \$25

Pagemaker files can be supplied - additional charge - \$35

CorelDraw files can be supplied - additional charge - \$30

Sponsor panels are 1.5" high reversed strips along the bottom of article pages. Graphics can protrude above the boundary. Call for us to fax you a sample.

All ad dimensions are directly proportional to page size.

Page Size - Letter (US)

CIRCULATION - CALL FOR UP TO DATE FIGURES.

Greater than 5000 readers as at last reader survey (August '94)

Approximately 100 New subscribers per month

**FOR MORE INFORMATION or a sample issue
Email Stephen Moignard on 76711,1243**

