Desaware

Desaware Inc. 1100 East Hamilton Ave, Suite 4, Campbell, CA 95008 USA
(408) 377-4770 fax: (408) 371-3530,
Conact support@desaware.com or Compuserve: 74431,3534
or visit *http://www.desaware.com*

VersionStamper 5.0

To order VersionStamper from this CD - please contact ComponentSource

U.S. Edition
Phone: (888) 850-9911, fax: (888)-850-9922
3605 Sandy Plains Road, Suite 240-237
Marietta, GA 30066

European Edition
Phone: +44 (0) 118 958 1111
27-37 Vachel Road
Reading, RG1 1NY, UNITED KINGDOM

Has this ever happened to you?

A customer calls - your program doesn't run - or it's behaving strangely. After hours of valuable support time it turns out that a DLL, OCX or VBX component had been accidently overwritten with an older component, or was somehow misregistered, or was missing, or an incompatible version of the component had been registered by another application.

Wouldn't it be nice if:

- You could detect those incompatibilities without wasting hours on the phone?
- You could determine instantly exactly what components your end-user has?
- Your application could automatically perform its own component verification?
- Your application could send you Email when it detects an incompatible component?
- Your application could verify its components against a list on your web or FTP site on your corporate intranet, or over the Internet?
- Your application could automatically install updated software components from your corporate FTP or web site, or over the Internet?
- Your developers could quickly determine all of the component dependencies of their applications or components.

We tell our customers that VersionStamper wil pay for itself the very first time a customer calls with component incompatibility problems. Our customers tell us that we're right.

And you risk nothing by finding out for yourself.

VersionStamper **-**- The best way to resolve file incompatibilities, now with Internet/Intranet update capabilities.

For your information:

In addition to the normal marketing material you would expect, we're pleased to offer some special bonuses with this online catalog:

The Desaware Visual Basic Bulletin - Access recent issues of our advanced technical newsletter.

Visual Basic 5.0 Programmer's Guide to the Win32 API Learn to use the 32 bit Windows API from Visual Basic.

Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed - Go beyond the Microsoft documentation to learn the inside scoop on developing ActiveX code components, ActiveX controls and ActiveX documents.

White Paper: Persistence of Data

A 5000 word feature article by Daniel Appleman, author of the Visual Basic Programmer's Guide to the Win32 API, on the various techniques available to Visual Basic programmers for storing data - and the tradeoffs involved on choosing the correct solution for particular applications.

Technical Notes

www.desaware.com - Your online source for updated information about our products and the Visual Basic Programmer's Guide to the Win32 API. Not to mention a selection of advanced Visual Basic tips and techniques.

Desaware maintains a list server which we use to let folks know about new issues of our newsletter and new products. (Don't worry though, we use it very infrequently and keep it short). Send a message to Listserve@desaware.com with the word "Subscribe" in the subject line.

VersionStamper

You get a technical support call....

"Your program has stopped working!"

Did someone install an older version of a DLL, VBX or ActiveX/OCX control required by your application?

Is another program using an incompatible version of a DLL, VBX or ActiveX/OCX control that it loaded from a private directory, preventing yours from loading?

Is an incompatible version of a DLL, VBX or ActiveX/OCX control present somewhere in the PATH and being loaded before the one that is required?

Did the user delete a component that is needed?

Did the user reinstall your application using an older release disk - copying over your most recent files because your VB application does not contain version information?

If only your application could tell you what was wrong....

Now it can....

Read about new Internet/Intranet Features in Version 5.0

Solving the Installation Problem.

Make no mistake - there is no simple "fix" to the problem of safely distributing Visual Basic applications. However, if you have complete information about the incompatibilities that occur and the versions of both components and VB executables, it is possible to quickly determine exactly what problems exist. Once you have this information, it becomes easy to find the correct solution.

VersionStamper allows you to easily embed into your Visual Basic executable or other OLE container information about all of the dynamic link libraries and custom controls used by that application, along with required current versions, file date and time, and your choice of warning conditions. The **VersionStamper** OLE custom control can use this information to check the versions of the actual modules that are available on the target system.

dwvstp??.ocx - The 16 and 32 bit ActiveX control includes extensive dialog box support for embedding version information for selected DLLs, OCXs, VBXs and other files. Each control can embed two file lists so you can use a single control to create dual platform executables. It also has the ability to scan Visual Basic -based projects to automatically create a list of DLLs, OCXs, and VBXs required by the application. The OLE version even allows you to scan for files not specified in your file list during run-time. The run-time distributable control is royalty free. **These controls are fully compatible with Visual Basic 4.0, 5.0 and other true ActiveX Control containers!**

VersionStamper 5.0 is also capable of reading file lists off of an Internet/Intranet web or ftp site so that your end user can verify that they are using the very latest components!

If you are still working with Visual Basic 3.0 -

The first, and most obvious need is to embed a standard Windows version resource into executables created by Visual Basic. This will make it possible for any standard Windows installation program to correctly handle your Visual Basic programs and prevent the accidental overwriting of later versions of your executables.

VersionStamper contains the dwvstamp.vbx custom control which makes the process of embedding a version resource into your executable completely automatic. The version information is entered by setting the control's properties at design time just as you would set the properties for any custom control. This information will be converted into a standard version resource automatically during the compilation process.

Solving the Distribution Problem.

The key to eliminating problems relating to the distribution of Visual Basic applications is to be able to determine, quickly, whether all of the components required by the application are, in fact, present in the runtime environment.

Embedding component version information into your application.

The VersionStamper control makes it possible to embed into the executable a complete list of every DLL, VBX or other software component that is needed by your program. This embedded component information includes the required version information for each component. You can also specify the conditions under which a warning will be triggered.

VersionStamper is able to automatically scan a project for a list of the custom controls used by the program and a list of all DLLs referenced by the Declare statements in the project. VersionStamper 5.0 has the ability to determine indirect dependencies - DLL's or OCX's that depend on other dynamic link libraries.

VersionStamper is also able to download a list of required components and version numbers from a web or ftp site.

Detecting component incompatibilities at runtime.

Since an executable stamped by VersionStamper "knows" exactly what components it requires at runtime, it is an easy matter for it to check if all of those components are available and up to date. This comparison process is also built into the VersionStamper custom control.

VersionStamper provides a great deal of flexibility in terms of how you handle the scanning and reporting process. Beginning programmers may simply choose to add the standard Vervrfy.frm form or vervrfy2.frm form and associated modules to their application to provide a default verification check at load time or under user control. The OLE version even allows you to scan for files not specified in your file list during run-time. The run-time distributable control is royalty free. **These controls are fully compatible with Visual Basic 4.0, 5.0 and other true ActiveX Control containers!** More advanced programmers may choose to modify the code to create their own custom report. You might also provide the user with a step-by-step procedure on how to fix certain problems, or perhaps a request that they call your customer support line.

It is possible for the software components required by an application to be so old that the program will not run at all. For these cases VersionStamper includes two sample "Rescue" programs which are able to compare the component requirements of an executable with the runtime environment without actually loading that executable. These rescue programs include complete source code and can be fully customized to suit your own needs.

New!!! Self-Updating Applications.

What happens when VersionStamper discovers and incompatible or obsolete component? With version 5, VersionStamper adds the ability to automatically download and install the updated component from an FTP site over the Internet or corporate Intranet!

Additional Tools and Information

VersionStamper includes a program called VerInfo.exe which is able to report on the version information of any executable that contains a version resource. It is also able to report on the embedded component information added by the VersionStamper custom control.

This program also includes **full source code**. Which brings us to one of the final features of this package, a feature that we feel is one of the most important.

One of our goals at Desaware is to not just provide "canned solutions" for programmers but to also help educate programmers to take advantage of all of the technology that is available to Visual Basic programmers. Following this philosophy, you will find a fair amount of technical material in this manual and a generous amount of Visual Basic source code as well. We will also cover some of the reasons why certain things work the way they do and how they influenced the design of VersionStamper.

Some additional features of VersionStamper are:

- **VerResQ.exe -** A Visual Basic-based program that analyzes any Visual Basic-based executable stamped by **VersionStamper**. It instantly detects any incompatibilities that exist between the requested files and those actually present on disk. Includes source code so that it can be customized for distribution with your application.
- **VerSplsh.mak** - This sample project demonstrates using a standalone executable to perform file verification for your main application. It displays a splash screen during file verification. If no conflicts were found, it Shells your application, otherwise a form is displayed notifying the user of the conflicts. This method will allow you to identify any incompatibilities that exist - even if the executable cannot be loaded. Designed to be easily modified so that you can create your own splash screen shell program to distribute with your application.
- **VsRTDemo.mak** - This sample project demonstrates a new feature of **VersionStamper** (OLE control only) - how to scan files dynamically during run-time.
- **Technical Notes** include a complete discussion of the Windows-based system of version stamping and the embedding techniques used by **VersionStamper**. It also includes detailed suggestions and examples for improving the Visual Basic setup toolkit. It also includes an extensive discussion of platform dependent issues including Windows 95 and Windows NT.

But fear not - this product was designed to be used by absolute beginners as well. And for those who just need a quick solution to the distribution problems discussed earlier, feel free to skip forward to the Quick Start section which will walk you through the process of embedding component information and adding a runtime conflict report (and embedding a version resource for VB3.0) in a few easy steps.

Visual Basic 5.0 Programmer's Guide to the Win32 API

Author - Daniel Appleman

The original Visual Basic Programmer's Guide to the Windows API has helped tens of thousands of Visual Basic programmers incorporate advanced Windows API programming into their Visual Basic application. The new Win32 32-bit API for Windows NT and Windows 95 is even more powerful, not to mention considerably larger. Daniel Appleman's new book on the 32 bit API for Visual Basic programmers goes beyond just covering the new API functions. It includes more extensive explanations, more sample programs, and more subjects. Also extensive information on porting from 16 bit applications and writing code that will run on both 16 and 32 systems. Now updated for Windows NT 4.0 and Visual Basic 5.0 (while remaining fully compatible with Visual Basic 4.0). Plus: Over 1500 pages and a full text searchable electronic edition included on the CD-ROM! The book includes a portion of the Desaware API Class Library. Build on your knowledge of Visual Basic to become a Windows expert by learning to take advantage of the hundreds of functions contained in the Windows API. No knowledge of 'C' or prior Windows knowledge is required. Written by the president of Desaware, it covers virtually every aspect of Windows and includes extensive sample code. Published by Ziff-Davis Press. ISBN: 1-56276-446-2.

News Flash !!!

Upgrade Program for owners of the PC Magazine Visual Basic Programmer's Guide to the Win32 API!

The Visual Basic Programmer's Guide to the Win32 API posed an interesting dilemma for both the publisher and myself. The following facts became evident as we discussed how to proceed with the update:

While some changes were definitely needed in order to update the book for Visual Basic 5.0 and NT 4.0, the changes were relatively minor.

- A sad fact of marketing is that book stores will quickly remove books for previous versions of Visual Basic from the shelves as soon as VB5 appears.
- The publisher discovered that they had underpriced the book - you see, all of their forecasts were based on a 1000 page book, and I delivered a 1500 page book.
- I wanted to make sure the programmers who bought the VB4 edition of the book would not feel that they were being manipulated into buying a new version that did not have significant new material.

After extended discussion, we agreed that the second edition of the book would be released under a slightly modified title at a higher price. At the same time I negotiated a CD-ROM update program at a substantial discount for current owners of the book. Keep in mind that the CD-ROM contains the full text of the book, so this does provide a way to obtain all of the VB5 compatible source code, along with the very latest text of the book (including a couple of new chapters that are only on the CD-ROM).

The CD-ROM is available only directly from Desaware for $24.99 plus shipping & handling. Please visit our web site at www.desaware.com for details.

Dan

Developing ActiveX Components with Visual Basic 5.0: A Guide to the perplexed.

Author - Daniel Appleman

This book is **NOT** the ultimate Bible for developing ActiveX components with Visual Basic 5.0. The Bible already exists - it's the documentation provided by Microsoft with VB.

This book is the commentary.

This book goes beyond the Microsoft documentation to provide additional illustration, interpretation, examples, personal observations by the author, and advice to help you become an expert on ActiveX Component development with Visual Basic.

Part 1 discusses core technologies. Find out the truth about ActiveX (is it real or is it just a marketing gimmick). Learn the principles of object oriented programming from a VB perspective. Find out what COM is all about and learn all about the new VB5 features for polymorphism.

Part 2 discusses code components. See the myriad of project options. Learn about raising events, about collections and object lifetime. Discover the truth about multithreading.

Part 3 discusses ActiveX controls. From simple constituent based controls to user drawn controls to subclassed controls with custom window message handlers, you'll not only learn how to create different types of controls, but the design tradeoffs as well. You'll find out the limitations of Visual Basic based controls as well as their advantages.

Part 4 discusses ActiveX documents - based on the still unproved assumption that somebody out there may find them useful.

Part 5 concludes with a discussion of distribution and licensing issues, plus whatever else the author decided to throw in at the last minute.

This is the book that the author wished he had while reading the Visual Basic manuals. He had to work almost half a year to get it.

Published by Ziff-Davis Press. ISBN: 1-56276-510-8.

Visual Basic Programmer's Guide to the 16 Bit Windows API

Build on your knowledge of Visual Basic to become a Windows expert by learning to take advantage of the hundreds of functions contained in the Windows API. No knowledge of 'C' or prior Windows knowledge is required. Written by the president of Desaware, it covers virtually every aspect of Windows and includes extensive sample code. Over 1000 pages plus disk. Published by Ziff-Davis Press. ISBN: 1-56276-073-4.

Technical Notes

Desaware's products are designed to be used by professionals. This means that we do our best to make them efficient in terms of resource and memory use. Some of the most important features of these products are often "behind the scenes". The purpose of this paper is to introduce you to some of these hidden features.

The art of subclassing:

You would be surprised how many people have asked me recently if SpyWorks was obsolete. After all, since it is possible to subclass a window using Visual Basic alone, SpyWorks becomes superfluous, right?

Think again.

Let's avoid for a moment the fact that subclassing is but one of the many capabilities provided by SpyWorks - especially in the new SpyWorks 5.0 edition. Let's consider just subclassing.

The first think that you will find when you try to subclass a window is that your application will suddenly become difficult, if not impossible, to debug. The instant your program enters break mode, your subclassing code will stop executing, meaning that the window being subclassed will no longer work correctly. This can easily destabilize your debugging environment and may even cause it to crash. This means that even under VB5 you will almost always want to implement subclassing within a DLL (even if it is written in Visual Basic).

The next thing you will find is that while VB subclassing works within a process, any attempt to subclass a window in another application will invariably lead to a memory exception. Cross task subclassing is notoriously difficult to do, and I do not believe VB5 is a sufficiently low level tool to do so safely.

You should also be aware that subclassing is a simple idea in concept, but can be very tricky in practice - especially in an environment such as Visual Basic where a form or control may be subclassed several times by different 3rd party tools. SpyWorks has been designed with this in mind, and uses a number of advanced techniques to cooperate with and protect itself from other tools and applications.

Finally, subclassing a window by its very nature implies that the subclassing code will receive and process every message received by the window being subclassed. The performance impact of subclassing itself is very small - however neither we, nor Microsoft guarantees that it is safe to fire a VB event during every Windows message. We know for a fact that there are many operations that are not allowed during certain types of message processing. In order to provide the best possible performance, all of Desaware's subclassers only trigger VB events for those messages that you specify. This significantly reduces the chance of accidental errors, plus it reduces the amount of VB code required for message processing.

Does this mean that you should use Desaware's advanced subclassing ActiveX controls for all of your Visual Basic applications? Certainly not!

You see, now that Visual Basic allows you to write a subclasser using Visual Basic alone, it goes without saying that we did so. And we wrote a good one - a subclasser that provides low level filtering, that correctly coexists with other subclassers or controls that use subclassing (even those that behave poorly), and that uses early binding to provide the highest possible performance with the lowest possible overhead. And while we were at it, we wrote an thread specific hook control to go with it.

These components are included in the new version of SpyWorks - *with full source code!*

The way I see it, now that Visual Basic lets you use subclassing, it's our job to show you how to do it right.

And for those cases where you need to subclass or hook other tasks, the advanced SpyWorks controls remain the most robust, popular and reliable solutions in the business.

The VersionStamper controls - Automation or Full control + efficiency.

Visual Basic promotes the use of software components - OCX's, DLL's and so on. But while Windows

provides good technical support for use of components, it is pretty clear that its designers never considered the real world problems that occur on systems with multiple versions of shared components. We ourselves have spent plenty of time on the phone chasing down bugs that turned out to be old versions of controls that somehow "reappeared" on our customer's systems.

VersionStamper is our way of providing a tool to solve these problems when they occur. We had three major design criteria. First, we wanted it to be easy to use for those who wanted a quick and simple solution to these distribution problems. If you accept the control's defaults, all you need to do is select embedded files, type in your version numbers, and add a module and form to your application that contains our default report generator (assuming you want that capability).

But we also wanted to provide flexibility for serious programmers. As such, all conflict resolution is handled through Visual Basic events, allowing you to take complete control over the way any problems are handled. The package includes plenty of source code (all of the Visual Basic utilities include source code).

Finally, we wanted the control to be efficient. As such, we placed all code used at design time into a separate design time DLL that is not distributed with your application.

Our Technical Vision - a note from the president:

It is my belief that Visual Basic represents the dominant paradigm for programming now and for the foreseeable future. Visual Basic itself is an impressive implementation of that paradigm, and I expect that it will continue to be one of the dominant platforms for Visual programming.

One reason for this is that it is possible to use Visual Basic to write Windows applications without compromising on the power of traditional C and SDK based techniques. I've seen a great many VB programmers get frustrated by their inability to do something in VB and blame the environment itself. My primary goal with Desaware (aside from keeping a roof over my head), is to help VB programmers break through these walls both through education, and by providing innovative development tools.

I think we've made a good first step in this direction. My first book, "PC Magazine's Visual Basic Programmer's Guide to the Windows API" was very well received as a "Windows SDK" for the Visual Basic programmer (and continues to sell well for those doing 16 bit programming). SpyWorks provides the low level hooks and debugging tools required to do the heretofore "impossible" in VB. VersionStamper is the first product to seriously address the problems of distributing applications made from a large number of components. The Visual Basic Programmer's Guide to the Win32 API and SpyWorks 5.0 extend this philosophy into the 32 bit world. My new book "Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed" extends our focus from Windows API techniques to include OLE and ActiveX techniques.

In the future Desaware will continue with this approach. You won't be seeing large libraries of controls or high end solutions - there are many vendors providing excellent tools of that type. Instead watch for development tools that enable you to develop your own solutions, for the debugging tools needed to learn enough to create those solutions, and for useful and interesting controls or applications that include full source such as those in our new ActiveX Gallimaufry product - plus, of course, newsletters and a web site that will build on the knowledge base of the Visual Basic community at large (and of course, Desaware's customers in particular).

Thank you for your support in the past, and I look forward to working with you in the future.

Daniel Appleman

Persistence of Data - A Technical White Paper

When should your VB programs use Databases, the Registry, .INI files, or Disk files to save information? And what is OLE Structured Storage? Find out in this new article by the author of the Visual Basic Programmer's Guide to the Win32 API.

by Daniel Appleman

The author may be reached at dan@desaware.com or on Compuserve at 70303,2252 or Internet: 70303.2252@compuserve.com

The following technology white paper is based on a talk that I presented at the Silicon Valley Visual Basic User's Group in January 1996. It is being distributed in an effort to educate Visual Basic programmers regarding available techniques for implementing data storage in their applications. The paper focuses on matching an appropriate data storage technology to the needs of individual applications, and discusses the advantages and disadvantages of each approach. The end of this article does discuss a Desaware product (the company for which I work), however you will find that the bulk of the article is an objective and technical discussion on storage techniques that should prove valuable to every Visual Basic programmer. By reading this article you will gain:

- An understanding of many of the tradeoffs involved in data storage technologies including initialization files, the system registry, disk files, database systems and structured storage systems.
- An excellent understanding of OLE structured storage as it relates to application design - including where its use is and is not appropriate. Also, you will see how OLE structured storage fits into Microsoft's OLE strategy.

Introduction

OLE structured storage is one of the most exciting technologies to come along in quite a while. It will dramatically change the way you work with files. It is.....

Wait. Isn't this the way most articles begin when describing a cool technology? They focus on the features and capabilities of the tool or technique being described. But is this really the information that most programmers need?

You see, most programmers (myself included) tend to fall in love with technology. We hear about a cool new tool or technique, and it becomes, for a while, the greatest thing that we've ever heard of. When I was studying computer science I remember being introduced to a new language every week or two, and each one instantly became the "best" language in my repertoire - the language that I could surely use to perform any task more efficiently and quickly than any other. And it would remain the "best" language until the next one came along.

This is something that I think is common among many programmers and engineers. We become passionate about technology. We tend to love our work. Our preferences of languages, tools and machines are almost religious in their intensity (in fact, I dare say that there are programmers who are considerably more passionate about their programming tools than their religion).

So before discussing this "cool new tool" called OLE structured storage, I think it is important that we pause for a moment and place it in perspective.

You see, as programmers our real job is not to fall in love with technology. It is to solve problems using software. And as such, it is our professional responsibility to choose the tools that will best help us solve the problem at hand - not the one that we happen to be excited about at the moment, or the one

that the industry is busy hyping.

That said, here is the truth about OLE structured storage: It is a powerful technology that is highly suited to solving particular problems, and totally unsuitable for others. So, how does one determine which problems are right for this technology? Let us start with considering the entire class of problem at hand: that of persistence of data.

Persistence of Data

Imagine a program such as a word processor that was unable to save information from one session to the next. With such a program, you would have to type in a document from scratch every time you needed a printout - roughly akin to the way offices used typewriters just a few decades ago. Much of the value of computers comes from the ability of applications to store information - to persist their data. There are many technologies available for persisting data, and not all of them are appropriate for every situation.

There are three major reasons for persisting data in an application and one or all may be appropriate for any given program. They are as follows:

1. **Application configuration information**: Many applications save state information from one session to the next. It may remember user settings such as whether a backup should be generated for each file, or the positions of windows so that the application will restore itself to the same state that it was in when it was last closed. The application might save general configuration information as well as user specific configuration settings (for applications that support multiple users).
2. **Documents**: Many applications work on documents which are opened at the beginning of a session and saved or closed when the session is over. Typical documents are word processing documents, spreadsheets, pictures and so forth. Documents can often contain many types of objects. The key thing to remember about documents is that they generally correspond to a single disk file. This makes a document easy to manage as an individual unit. For example: it can be copied to a diskette, or sent via Email.
3. **Records**: Applications that need to keep track of multiple records which are similar to each other will typically use some sort of database storage mechanism. Databases support searching and filtering and are designed to be easily shared by multiple applications simultaneously.

Each of these reasons for storing data influences the choice of technology that can best solve the particular storage problem. But the choice of technology is not always obvious. Let's take a look at the major storage technologies in use under Windows and how appropriate each is for the problems listed above.

Private Initialization Files

Initialization files are an older technology that goes back to Window 2.0 (version 1.0 did not support private initialization files, only win.ini). While some of the newer Windows documentation suggests that one should always use the registry instead of initialization files, this is not necessarily the case. Initialization files maintain a number of advantages over the registry for some applications. They can be edited by hand using any text editor such as Notepad. They can be easily copied or saved using simple file commands. They are very easy to work with using a few simple API functions or Visual Basic commands.

Are initialization files appropriate for saving application configuration information? Yes - absolutely. A program's state can often be stored as a relatively small number of short string entries or numeric entries - exactly the kind of information that initialization files handle best. It would take more effort to handle user specific configuration information - each user would have to have their own section in the file, but it is still a viable solution.

Are initialization files appropriate for saving documents? Probably not. Their performance is not adequate for most documents. Each entry is limited in size, as is the entire file (depending on the operating system). They can only handle text data, limiting their flexibility for general use.

Are initialization files appropriate for saving record information? Certainly not, except perhaps for the most limited applications.

The System Registry

The system registry or registration database provides hierarchical storage in a central location on each system. The registry is quite limited under Windows 3.x, supporting only limited data types and only one value at each node in the hierarchy. Under Windows 95 and Windows NT, however, each key in the registry may have any number of named values associated with it, and those values can be one of a wide selection of available data types including raw binary data. Each value is generally under 2k bytes in length.

Is the registry appropriate for saving application configuration information? Absolutely. It provides a far more sophisticated structure for saving configuration information and you can easily save user specific information by simply giving each user their own node in the registry hierarchy. The registry is, however, harder for an end user to edit (which is either an advantage or disadvantage, depending on your application). Editing the registry always opens the risk that the user might accidentally make a change that could interfere with the correct operation of the system - since the registry, unlike private initialization files, also contains configuration information for the system itself. The registry is harder to access programmatically than private initialization files, requiring the use of more sophisticated API commands (though a simple set of Visual Basic commands are also available for basic registry operations).

Is the registry appropriate for saving documents? Certainly not. There is no clean mechanism for managing individual parts of the registry - how would you copy part of the registry to a disk file or send it through Email? The registry is, by its very nature, not document oriented. It is also strongly recommended that each value in the registry be kept relatively small.

Is the registry appropriate for saving record information? Not a chance - all the reasons discussed for documents apply here as well.

Independent Documents

One of the most common methods used by applications to persist data is to store the required information in document files on disk. There are a number of standard formats for disk files, for example: .BMP files (windows bitmap format) is a standard file format for bitmap images. RTF (rich text format) is a standard formatted text document format. In many cases, applications define their own private file formats - in fact, many standard formats originated in that manner. Whether a private or public file format is appropriate for a given application is one of the design choices that must be made by the developer.

With document files, the contents of the file are the responsibility of the application, as are all of the file operations. File operations are implemented using either API commands or a wide variety of Visual Basic commands that allow you to work with lines, records or binary data.

Are document files appropriate for saving application configuration information? Yes - after all, a private initialization file is, essentially, a document on disk. But document files are rarely used in this way. If the configuration information is simple, it is easier to use an initialization file. If the configuration information is complex or requires the persistence of data that cannot be easily represented as text, it is often easier to use the system registry.

Document files are easy to manage on disk. They can be copied, moved, sent via Email, and managed using standard file management software.

The task of reading and writing document files can range from very simple to extremely complex, depending on the type of document. A simple text document can be created by simply printing lines of text into the file, each one separated by a carriage-return line-feed combination. If, however, you wish to insert a line into the middle of the file, or remove a line from the file, you will generally need to write the entire file from scratch - there is no easy way to shift information in a disk file.

What if you wish to save many different types of information in the document? For example: a word processing file that contains both text and pictures? In that case you need to define a more complex file format that keeps track of where individual objects are located within the disk file. You might need to actually define an internal directory within the disk file that lists the various parts of the file and where they are located. In fact, if you think about it, you might reach a level of complexity in file organization that is used by database systems (which routinely track large numbers of arbitrarily sized objects within a disk file).

Which brings us to the subject of records. Is a document file appropriate for storing multiple records of

data that are similar to each other? The answer here is yes and no. Let's say your application uses a user defined type (data structure) internally to store data, and needs to persist an array of these structures. In this case, a document file can be an extremely efficient way to store this information. Visual Basic includes record I/O file operations that are fast and have low overhead.

The document approach towards storing records does, however, have a number of potential disadvantages. Insertion and deletion of individual records can be complex. If there is a chance that the structure format might change, your application will need to keep track of document versions and implement a scheme for updating documents from one version to the next. Use of variable length records requires a more complex storage scheme that includes some mechanism for finding the start and end location of each record. Also, any searching or filtering schemes need to be implemented by the application.

Records

In many cases you may need to persist groups of records, where the records are similar to each other in structure. These often take the form of tables of records, where each record contains the same fields. You may need to work with different types of records in the same file (multiple tables). In many cases, you will want the information to be accessible to multiple users or applications at once. Clearly, this is the type of information that is often stored using a database system. This might include use of a specialized database command set or the database controls and commands built into Visual Basic.

You may be thinking that this is a somewhat convoluted way of talking about databases. Why go to all of the trouble to differentiate between "saving records" and building a database? Because it is important to distinguish between the goal and the solution. The goal is to persist a certain type of data: records that are similar to each other. One solution is a database - but it is not the only solution.

You have already seen that it is possible to store records in a document file. However, if a database is clearly designed for record storage, why would you ever want to store records in a document file? You might want to do so if you do not need the database's searching and filtering capability, if you have little or no need to share the file and if the records are simple and of fixed length. In that case not only might the document approach be easier to program - it will probably be faster and will allow you to avoid the overhead of including a database engine with your application.

The case for application configuration information is easier to see. A database would almost always be extreme overkill for this type of data persistence. Even storage of per user application information is more easily accomplished using any of the other techniques.

What about documents? Is it appropriate to use a database to store document information - a word processing document, for example? It certainly can be done. You can imagine a complex database structure that includes one table for the lines of the document (perhaps with formatting information), and other tables that contain different types of objects, perhaps in blob format to hold images, video, sound, and so forth. These tables can be linked through the relational capabilities of the database.

This approach would work - but it can become quite complex - easily as complex as implementing the document using a proprietary format and performing your own disk operations. The database approach also can have a significant impact on performance and involves significant overhead.

The Missing Technology

All programmers have to deal with the problem of persisting data, and there is a tendency to jump straight to a solution without considering the problem in the context of the requirements of a particular software project. It is true, as you have seen, that application configuration information is usually best handled using either a private initialization file or the system registry. It is true that record storage is usually best handled by a database. It is true that simple documents are usually best handled by reading and writing a disk file directly. But you have also seen that each of the four technologies that we've discussed can be applied for any of these persistence tasks, and the obvious choice is not always the best.

It is also clear from the previous discussion that there is one type of persistence problem for which none of the technologies is ideal. What is the best way to store complex documents that contain different types of information? Implementing a complex file format is a great deal of work. And databases are designed for records that are similar to each other - using them to store arbitrary types

of information of arbitrary length is possible in many cases, but can be even more work.

Until recently, there has been no obvious solution to this problem available to Visual Basic programmers.

Before looking at OLE structured storage in the context of this problem, let's pause for a moment and consider the characteristics of the ideal complex document storage system - a system able to manage a complex document file. Assume for the moment, that the term "object" refers to any block of data that you care to define.

- The file must be able to hold a very large number of objects, of types defined by the application. Each of the objects must be of any arbitrary size, efficiently handled whether it is a few bytes or megabytes in size.
- It must be possible to insert and remove objects, or change their size or contents without needing to rewrite the entire file.
- It must be possible to write data into the space allocated for an object, without risk of interfering with other objects stored in the file.
- The storage system must maintain an internal table of information about the objects in the file, making it easy to find and reference any object. It would be even nicer if it were possible to organize these objects in some hierarchical manner of objects and sub-objects.

The complexity of implementing such a system on your own is substantial - and one of the reasons that it is so tempting to look for alternatives to creating a private file format. At the same time, this description might sound familiar - in fact, a storage system that meets these criteria is present on every Windows and DOS based system.

It is the file system.

Think for a moment of files as objects. A file system can handle many thousands of objects. Each file can range from zero bytes to gigabytes in size. You can certainly add and remove files without fear of one file interfering with the next. And writing into one file cannot corrupt any other. Files are named, and can be organized in a hierarchy of directories.

In fact, you will find some applications that solve the problem of complex document storage by using the file system - dividing their documents into multiple disk files. An example that I worked with recently is Corel Ventura Publisher, in which one publication can consist of multiple chapters each of which contains links to documents of different types. A single publication can thus be made up of hundreds of files, which are typically organized in one or more directories.

The disadvantages of this approach are clear when you try to copy a document to a floppy disk, to another directory, or to send it via Email. Copying the publication requires a special utility that can search out all of the linked files and copy them to the correct destination. If even one file is missing, the entire publication can fail to load at worst, have missing information at best.

Which brings us to OLE Structured Storage.

OLE Structured Storage

It sometimes seems that part of the challenge of being a Windows developer (whether you use Visual Basic or another language) is coping with the multitude of new terms and acronyms that are generated with great frequency from Redmond. OLE Structured Storage is one of these.

Fortunately, this is where the somewhat long introductory material that you have read up to now serves its purpose. Because it makes it possible to understand exactly what OLE Structured Storage is and why it exists and even how it might fit into your own development plans.

Think again, for a moment, about a file system - both its advantages and disadvantages, and how it might be used to address the problem of creating complex documents. Consider then, that a file system exists on a hard disk drive - a large area that is able to store information. What would happen if, instead, you could place an entire file system within a single disk file? Suddenly you would have all of the advantages of a file system, with none of the disadvantages. Since the entire file system is within a single disk file, you can copy, move and Email all of the objects for that file system by simply copying, moving or Emailing the file that contains the file system.

OLE Structure Storage is a technology that places an entire file system within a single disk file. Nothing more and nothing less. And understanding that this is what OLE structured storage is, you know nearly

as much about it as any of the experts.

Terminology

In a file system you have directories and files. Each directory has a top level root directory. Each directory can contain multiple subdirectories and any number of files.

In OLE Structured Storage, the term *Storage* is used in place of directory, and the term *Stream* is used in place of file. When working with OLE Structured Storage Files, you start with a top level storage (*root storage)*. This storage can contain additional sub storages and streams. Streams can be created, opened and closed just like files. You can write into streams without worrying that one might interfere with another. OLE Structured Storage takes care of allocating space for streams and storages, and managing space allocation within the disk file.

OLE Structured Storage also defines a name for a disk file that is managed by this technology. These files are called *Compound Documents*.

Why is Structured Storage a part of OLE?

Structured Storage accurately describes what this compound document technology does, but why is this part of OLE? - the ‘‘Object Linking and Embedding’’ standard from Microsoft?

Part of this is the fact that OLE is in fact a collection of technologies - a sort of grab back under which many of the new developments for Windows falls. But part of it has to do with one of the major goals of OLE - the ability for users to deal with documents instead of applications.

Consider what happens when you embed an Excel spreadsheet into a Word document. When you click on the spreadsheet part of the document, the Excel user interface appears within Word - a technology called ‘‘in-place editing’’. The document is a word document, yet clearly, Word is able to somehow store and manipulate the Excel spreadsheet object that is somehow embedded into the Word file. Or can it?

In fact, Word has no idea what data is contained in the Excel spreadsheet or how to work with it. What actually happens is that the Word file is a compound document. One of the storages in the document is a place where objects foreign to Word are stored including, in this case, the Excel spreadsheet. Each object is stored in its own stream. When you save the word document, Word does not know how to store the spreadsheet object, but it does not need to - all it has to do is pass a handle to the stream to Excel. Excel then saves the spreadsheet data into the stream.

When you reload the document, Word looks at the objects in its object pool. Each of the streams starts with a GUID - a universal identifier. Word looks at that identifier and calls the OLE libraries asking it to create an object based on that stream information. OLE looks through the system registry, sees that it is an Excel spreadsheet object, then loads those parts of Excel that are necessary to display the object and allow the user to edit it.

As you can see, OLE Structured storage fills the critical need of allowing an application to store objects even though it does not know their internal format - it can simply pass a stream handle to a server that knows how to work with that object. You can use this same technique in Visual Basic in conjunction with the OLE container control to store foreign objects inside your own compound document files.

A quick look with a compound document browser shows that OLE Structured Storage is used by many Microsoft Applications including most of the Microsoft Office applications.

Should you use a Compound Document or a Database?

I’ve been asked this question any number of times. In fact, a large part of the purpose of this article is to dispel this question. Note that I say dispel - not answer. The question can’t be answered because it is intimately tied to the design of each individual application.

If your application is storing multiple groups of records, where all of the records in a group are similar to each other, and requires sophisticated search and filtering capabilities and support for the definition of relationships between records in different groups (tables), then you probably need a database.

If you wish to create a private document format that contains arbitrary types of data of different sizes, and perhaps objects from other applications, Structured Storage might be a better choice.

In either case, the problem is first to determine the needs of your particular application. Once the

requirements are defined, the choice of technology will, in most cases, be clear.

Private vs. Public Information

While the contents of any particular stream in a compound document is defined entirely by the application or object that it contains, the management of streams and storages within the document is the responsibility of OLE itself. This means that it is possible for any application to browse through the structure and contents of the document - just as any application can open and read a disk file.

One of the nice results of this characteristic of compound document files is that you can mix private and public data within a file. You can, for example, place a plain text description in a stream with a given name that any application can read.

Microsoft has actually defined a standard properties format stream called the ‘‘SummaryInformation’’ field. This stream contains standard summary information such as the document Title, Author, Comments and so forth. This means that any application can read this summary information. You can use it in your own documents, or use it to obtain information about other documents that use this feature.

Self Persisting Classes

You’ve seen how OLE Structured Storage allows each application to be responsible for its own objects within a single document. You can use the same technique within your own application by creating self persisting classes.

Say you create an application that contains a number of different classes and you wish to persist collections of objects from each class. One technique involves creating high level routines for saving and loading all of the objects. The disadvantage of this approach is that any changes to a class require that you modify the high level routines. This includes adding new classes, or changing the formats that a class uses to save its data. This kind of centralized approach is necessary in most cases where you are creating a private file format that you are managing yourself.

OLE structured storage makes possible a better approach in which each class knows how to transfer its data to and from a stream. In this case, all the high level routine has to do is call a ‘‘Save’’ or ‘‘Load’’ method for each instance of each class. If a class needs to change its data format, it can do so at will (assuming it maintains the ability to load the older format). The high level persistence routines do not need to be modified in any way.

Use of self-persisting classes can improve the long term reliability of an application by making it less likely that the persistence operation of one class might interfere with those of others used by the application - even when the code or data formats for individual classes are modified.

Other capabilities of OLE Structured Storage

Compound document files support sharing - individual storages can be opened for exclusive or shared access, in much the same way as disk files support sharing.

OLE Structured Storage also supports transactioning. You can open a storage in transacted mode. After modifying the contents of the storage you then have the choice of committing those changes or reverting to the original contents. This can be ideal for implementing features such as incremental Undo operations.

OLE Structured Storage and Visual Basic

There is just one catch that exists when it comes to using OLE Structured Storage technology from Visual Basic. That is that Visual Basic does not support this technology. OLE Structured Storage is not implemented using API calls but rather OLE interfaces, a system not supported in VB.

Fortunately, you can access all of the capabilities of OLE structured storage from Visual Basic using a third party package called ‘‘StorageTools’’ from Desaware. This package includes 16 and 32 bit OLE controls that support both structured storage and easy access to the system registry.

The next page includes a more detailed description of StorageTools and its capabilities. For further information contact:

Desaware Incorporated
1100 East Hamilton Ave, Suite 4

Campbell, CA 95008
(408) 377-4770. Fax: (408) 371-3530.
Compuserve 74431,3534 or 74431.3534@compuserve.com

Desaware

Desaware Inc. 1100 East Hamilton Ave, Suite 4, Campbell, CA 95008
(408) 377-4770 fax: (408) 371-3530, Compuserve: 74431,3534

StorageTools™ Version 1.0

The ultimate data storage and file manipulation toolkit!
Change the way you work with files and data forever!

You may have heard about OLE 2.0 with its ability to control applications and embed objects from one application into another. But there's another feature built into OLE 2.0 that until now was not accessible to Visual Basic programmers. It's called "Structured Storage" and StorageTools is your key to this powerful technology.

Structured Storage allows you to create files that organize information easily. Blocks of data can be placed in hierarchical form, just as files are placed in directories. It's like having an entire file system in each file. You can also create and work with compound documents - a new standard for document storage.

OLE 2.0 takes care of allocating and freeing space within a file, so just as you need not concern yourself with the physical placement of files on disk, you can disregard the actual location of data in the file. Plus: with its support for transactioning you can easily implement undo operations and incremental saves in your application.

StorageTools allows you to take advantage of the same file storage system used by Microsoft's own applications. Includes sample programs (with source) that let you examine the structure of any OLE 2.0 based file so you can see exactly how they do it!

StorageTools Storage Control:

The Storage control uses Visual Basic OLE Objects to represent storages and streams, creating a familiar interface for manipulating those elements using familiar "Get" and "Put" style commands. Within the Structured Storage standard are a number of functions that let you upgrade your file handling procedures easily: you can detect whether files are in Structured Storage format or not, and you can read any normal file as if it was a Structured Storage file. The Storage control also adds a number of features specifically for Visual Basic programmers. It lets you read and write information in any of the styles Visual Basic uses, such as "Sequential", "Binary" or "Random", including arrays.

StorageTools is the key to unlocking the following Structured Storage capabilities:

- Utilize a powerful new method for organizing information within a file or compound document.
- Buffered file modifications allow you to easily undo modifications to a file using Transactioning.
- The ability to create a compound document in a memory buffer instead of a disk file
- Full support for standard SummaryInformation fields lets you expose general document information while still maintaining a proprietary internal data format.
- Ability to look inside any compound document file including those generated by Microsoft's Office applications such as Word and Excel.
- Includes a compound document file browser with complete Visual Basic source code.
- You can organize information in memory exactly like you organize information on disk. Only one function call is required to copy information between a Storage in memory and a Storage on disk.
- Reduce disk access time by only saving the parts of the document that have changed.
- All disk space allocation and organization are done for you.

- **And more....**

StorageTools Registry Control:

The Registry Control makes it easy to access the Registry and the Registration Database. It provides all the power of Windows API calls and more. Keys within the Registry are accessed like directories. Values are converted to data types compatible with Visual Basic. Powerful search capabilities are included, allowing you not to only search among keys and value names, but within the value data itself. The documentation includes descriptions of areas in the Registry that might be useful and how to access them.

Using StorageTools - the Registry becomes the key to transforming your Visual Basic program into a truly professional application. Here are just some of the thing you can do:

- Register the filename extension to your documents, so your program is automatically launched when a user activates one.
- Create user-specific and machine-specific configuration settings, dramatically improving your user interface.
- Expose DDE links to your program.
- Interface with Windows 95: set icons, register an uninstaller, update reference counts for DLL's you use, and much more.
- Expose your program's version information.
- Read and modify system settings.
- **And more....**

StorageTools includes 16 & 32 bit OLE controls, documentation and sample code. *only* $129.

Desaware prides itself on products that address real-world programming issues. Desaware develops advanced tools that educate and expand the limits of software development. Desaware is the developer of SpyWorks, VersionStamper, StorageTools, The Custom Control Factory, CCF Cursors and The Common Dialog Toolkit. With the transition to 32 bit OLE controls, and new operating systems, these products reflect our commitment to remaining at the forefront of this new technology.

For further information contact:
Desaware Incorporated
1100 East Hamilton Ave, Suite 4
Campbell, CA 95008
(408) 377-4770. Fax: (408) 371-3530.
support@desaware.com or Compuserve 74431,3534

Final Comments

This article runs about 5000 words (not counting the product information). Normally, I would sell an article of this type to one of the major magazines. This article is an experiment to see if electronic media has become widespread enough to justify writing feature articles for direct distribution. My hope is that enough people will be interested enough in the article to read the product information that follows, and that the sales that result will offset the loss resulting from not selling the article through traditional channels.

I would appreciate any feedback you may have regarding this approach.

Daniel Appleman

The Desaware Visual Basic Bulletin

I hate junk mail.

No, that's not really fair. Sometimes something comes in that I actually find worthwhile. But the vast majority of the offers and flyers and so forth that I receive get trashed with barely a glance.

So when I started Desaware I faced a dilemma. I couldn't really afford to come up with large glossy advertising catalogs - especially knowing that most of them would probably just get thrown out. And from an environmental perspective, if I was going to mail out information, I wanted it to have value to the recipient. I wanted what I sent out to be the type of material that I myself would not mind receiving.

So I decided to do a newsletter. One that would have interesting technical information, some of which focused on our products, and some of which was just of general interest. And it would also include some percentage of product information, so that I wouldn't have to send anything else out.

To what degree I succeeded is for you to judge. The first four editions of the newsletter follow, pretty much as they originally appeared.

About the Newsletter Bonus Disks

About Desaware

Jump to Newsletter #6
Jump to Newsletter #5
Jump to Newsletter #4
Jump to Newsletter #3
Jump to Newsletter #2
Jump to Newsletter #1

The Desaware Visual Basic Bulletin

Volume 4 Issue 1

Introduction

Welcome to the Spring '97 issue of the Desaware Visual Basic Bulletin. A lot of people have been asking us how we would respond to Visual Basic 5.0. This was a critical question not just for us, but for every component add-on vendor. Will Visual Basic's new callback capability eliminate tools such as SpyWorks? Will Visual Basic's native code compiler eliminate the need for high performance components? Will Visual Basic's ability to create ActiveX controls eliminate the market for third party controls?

I believe that Visual Basic 5 will have a significant impact on the component market - and that it does pose a challenge to every component vendor. In deciding how Desaware would approach this technology transition, I decided that we should, above all, remain true to our roots and our vision: that Desaware will always provide the best source of advanced tools and information to Visual Basic programmers. That our products will continue to extend the reach of Visual Basic and at the same time educate Visual Basic programmers. In this issue you will read about some of the important new technologies that Visual Basic 5.0 makes possible. You'll discover that not only is SpyWorks NOT obsolete, but that we've taken advantage of Visual Basic's new capabilities to extend SpyWorks in stunning new directions. You'll find out about a new way for Visual Basic developers to use the internet. You'll see our response to the idea of Visual Basic controls replacing complex controls authored in C++ (our answer may surprise you). And by the way - I wrote another book; one that I think you'll find a worthy follow-up to the newly revised "Visual Basic 5.0 Programmer's Guide to the Win32 API". So sit back, relax, and see what we've been up to for the past six months.

— Daniel Appleman

Inside...

Is SpyWorks Obsolete

Visual Basic 5.0 includes a new operator called “AddressOf” which allows you to obtain the address in memory of a function in a standard Visual Basic module. This enables you to do three things in Visual Basic that, until now, have required a tool such as SpyWorks.

- Use API callback functions
- Do in-process subclassing
- Implement thread-specific hooks

The question is: does this operator make SpyWorks obsolete and unnecessary?

Well, one thing is clear right off - the SpyWorks callback control is not necessary under VB5 - in fact in the new edition of the “Visual Basic 5.0 Programmer’s Guide to the Win32 API” those examples that utilized callbacks were rewritten to take advantage of the AddressOf operator.

And the truth is, despite the fact that SpyWorks includes a great many features beyond the three noted above, one would suspect that VB5 would make SpyWorks less necessary than it was before.

That is, it would - if SpyWorks remained unchanged.

However, SpyWorks is not, and never has been, just a subclasser or hook control. SpyWorks is more of an attitude - the idea that Visual Basic should be able to do anything that other languages such as Visual C++ can. SpyWorks has always included a combination of controls, examples and information to let you take full control of your programming environment.

The first modification for SpyWorks 5 was simple: it’s very easy to get into trouble with subclassing (especially if you follow the directions for doing so included in the VB5 documentation). Subclassing is notoriously difficult to do safely and correctly. So Desaware’s initial task was to ensure that SpyWorks 5 includes quality subclassing and hook tools authored in Visual Basic - with full source code. As we see it, if VB5 allows a programmer to do something, it’s our job to demonstrate how to do it correctly.

The next thing we did, as we worked on our own VB5 software development, was identify areas where we could not accomplish a task using Visual Basic alone. We then added the necessary features into SpyWorks 5. The original SpyWorks pioneered the use of subclassing and hooks with the first commercial quality tools to assist the programmer in accomplishing these tasks in Visual Basic. Now SpyWorks 5 pioneers two new technologies for extending Visual Basic.

The Desaware ActiveX Extension DLL allows a programmer to override the behavior of OLE interfaces that are implemented by his/her objects, and to implement new interfaces - including non-dispatch interfaces. That description may not make a great deal of sense now, but you’ll find that it will make a big difference in developmental efforts - especially when it comes to creating ActiveX components and controls. This technology and ActiveX technology in general are examined in greater detail in the article “A Tale of Three Interfaces” located on page 8 of this issue.

The Desaware dwExport Library introduces our new Dynamic Export Technology TM<D> which allows you to export functions from your ActiveX DLL components using a few simple Visual Basic commands. Our system does not modify your compiled DLL in any way, so you don’t have to worry about possible incompatibilities. Now you can, for the first time, use Visual Basic to create Control Panel Applet extensions, export function libraries, true NT services (no SrvAny), ISAPI filters and more.

OLE hooks · Export functions · VB subclassers · Full featured cross-task subclassing and hooks · System hooks · System hotkeys · System macro recorder · Rollup windows · Status bars · Scrolling windows · Window creation management · Debugging tools with source · Tons of sample code and

more.

This is SpyWorks - the attitude that you truly can do anything in Visual Basic - and the tools and information to make it possible. So, is SpyWorks obsolete? You are invited to read the articles in this newsletter detailing these new technologies - then judge for yourself. We think that you'll agree that not only is SpyWorks not obsolete, but that it is more powerful, useful and educational than ever.

dwSpyVB.dll - Low level Windows Tools, With a Twist (and Source)

You've probably heard that the AddressOf operator allows you to subclass windows using Visual Basic 5. The VB5 documentation shows you how you can use the SetWindowLong function to replace the default window function for a window. Now we invite you to take the following short quiz on subclassing with Visual Basic:

What if the window is part of a different process?

Functions using the AddressOf operator must be in standard modules - how do you figure out which of your objects is associated with a particular message?

How do you make that association efficient? In other words - do so without string conversions or late bound method calls? And can you avoid using the somewhat inefficient collection object?

How do you implement low level message filtering to improve performance?

How do you manage subclassing in an environment where other controls may be subclassing the same windows that you are?

How do you correctly and safely clean up after yourself?

How do you debug your code in the Visual Basic environment when it uses subclassing or hooks?

How do you create and manage your own windows correctly?

All of these questions, and more, are answered in dwSpyVB.dll, a low level windows utility library written entirely in Visual Basic. In addition to the low level classes, dwSpyVB.dll contains a number of classes that implement a higher level of functionality - for example: a class that turns on the default scroll bars for any window. This is especially useful for creating user drawn ActiveX controls. A second class implements the PreTranslateMessage function which lets you override the default behavior of the tab and arrow keys. Yet another implements reliable timer operations - implementing an unlimited number of timers without the sixty five second limit of the standard timer.

Of course, you could figure out how to do all of these things on your own, and spend a few weeks (or more) writing your own tools - but with the dwSpyVB.dll library, not only do you get a complete, tested solution based upon techniques that Desaware has been perfecting for over five years - you get the source code as well - so you can add new features or provide your own long term support if you so choose.

By the way, if you try to subclass a window that is in a different process, or install a system hook using VB5, you will quickly run into trouble (specifically, a memory exception). You'll need to use the high end SpyWorks controls to perform these tasks.

DLL's and Subclassing

Here’s a hint from SpyWorks 5. Never, ever, do subclassing or hooks within your own application. When you subclass a window, it is crucial that all messages directed to the window be processed. Failing to do so can seriously interfere with the performance of your application, the Visual Basic environment and even your system - especially under Windows 95. The problem with subclassing within your application is that as soon as your application stops or enters break mode, the code that is supposed to handle incoming messages stops as well. This means that it becomes impossible to interactively debug applications that use this technique. You should always perform subclassing, hooks and custom window management within a compiled ActiveX DLL or control.

This limitation does not apply to most regular API callbacks. However you will run into trouble with API timer callbacks - especially if you use the multimedia timer.

The dwSpyVB.dll in-process window tools are designed to be used as a compiled ActiveX DLL. They also implement correct default operations, so if your application is stopped or in break mode, your program (and system) will continue to run correctly.

A Tale of 3 Interfaces: IPerPropertyBrowsing, IObjectSafety, IShellLink

Visual Basic has long been the easiest method to use in developing applications for Microsoft Windows. But with VB5, it is safe to say that Visual Basic is no longer “easy”. With this latest release, Visual Basic has been rebuilt from the ground up to be based internally on OLE (now called ActiveX) technology. This process began with VB4, but is now much more visible to the typical Visual Basic programmer. This fact is especially important for those developers who will be creating ActiveX components (and if you are not, you probably should be).

A complete and in depth explanation of OLE and ActiveX technology from the Visual Basic programmer’s point of view can be found in Dan Appleman’s new book: “Dan Appleman’s Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed” which should be available late March or early April. We will use the brief space we have available here to cover a few key points.

The first thing to remember is that an interface is a contract - it represents a set of functions that an object exposes to the world. An object can expose more than one interface - in which case the software using the object can “ask” for a particular interface to use.

Every interface in COM (the Component Object Model on which OLE is based), contains within it an interface called IUnknown which is used to keep track of the number of references to an object, and to request a different interface from an object. There is another type of interface, called an “automation” interface or “IDispatch” interface which allows an object to publish a list of properties and functions at runtime which can then be executed by programs using the object. Automation interfaces are very flexible, but do not support every possible type of variable - for example: they do not support user defined types. This is why a public Visual Basic class function cannot have a user defined type as a parameter.

When you create a form, class, ActiveX control or ActiveX document, you create an object that has a main interface which contains the functions and properties that you define for the object. You can also use the Implements statement to have your object support additional interfaces. But keep in mind that all of these interfaces are automation interfaces. You cannot use Visual Basic to create non-automation interfaces, and cannot add non-automation interfaces to your object using the Implements statement.

Visual Basic does, however, use non-dispatch interfaces internally. It must, because many of the features offered by Visual Basic including OLE Drag/Drop and ActiveX controls make extensive use of standard non-dispatch interfaces.

Let’s take a look at three different interfaces and what they mean to Visual Basic programmers:

Example 1: IPerPropertyBrowsing

Most Visual Basic programmers take the Visual Basic property page a little bit for granted. It is a bit amazing when you think about it. How can the property page populate itself with the properties for any ActiveX control? How does it know to display the word (Picture) or (Empty) for a picture control? How does it populate the drop down list box for enumerated properties?

It can do these things because ActiveX controls provide information about their properties through built interfaces. One of these interfaces, named IPerPropertyBrowsing, is responsible for allowing the Visual Basic property page (or any container) to retrieve a “display name” for the property - what it should display as the property value if the control wants to display something other than the value. It is also responsible for populating the drop down list for enumerated properties. This is an important issue for control developers.

To illustrate, you wish to display an enumerated property that can consist of three colors, red, green and blue. You could create an enumerated property:

```
Public Enum ColorList
   Red = 1
   Green = 2
   Blue = 3
End Enum
```

But this would be a bad idea. Why? Because it is very possible that another control will expose the same enumerated names with different numbers, thus leading to either a value conflict, or a subtle and hard to find bug. That is why you should always prefix the constant value with a product or company prefix that will help make your constants unique. For example:

```
Public Enum ColorList
   dwRed = 1
   dwGreen = 2
   dwBlue = 3
End Enum
```

This approach is better for programming and will cause the following to appear in the drop down list box when a developer uses the Visual Basic property window to configure your control.

1 - dwRed
2 - dwGreen
3 - dwBlue

However, the Visual Basic property window is the one place where you probably do not want the constant names - you might want it to read:
1 - Red pixel
2 - Green pixel
3 - Blue pixel

In order to do this, you need the ability to override Visual Basic's default implementation of the IPerPropertyBrowsing interface. SpyWorks 5 lets you do so, thus taking control of both the display name and the enumeration names for any properties in ActiveX controls that you develop.

Example 2: IObjectSafety

An important aspect of distributing your ActiveX controls over the internet and intranets relates to security. ActiveX controls have full access to all of the capabilities of your system, and thus have the ability to create havoc. Microsoft advocates a three tiered approach towards object safety for networks:

- The object must be safe.
- The object must have a way to inform the container to what degree it is actually safe.
- The container must have a way to identify who created the object and whether it has been tampered with.

What does it mean to make an object safe? It means that the object cannot cause harm to the end user's system. There are two issues to resolve: Is the object safe to load and initialize? Is it safe to program? Safe to initialize means that the ActiveX control can be placed on a web page and will be safe regardless of the initial property values that are loaded from the page. Safe to program, (also

known as “safe for scripting”) means that the object will do no harm to an end user’s system regardless of what the controlling program attempts to do. The control must be able to handle any possible function call and property setting safely. The degree of safety of an object depends entirely on the object’s author.

Microsoft defines two ways to mark a control as safe. The method utililized by VB5 is to add an entry to the registry section for the control describing it as safe to initialize or safe for scripting. This approach has two disadvantages: it requires that the control always meet the safety level marked, and it requires an extra registry look up each time the control is loaded. The second approach is to expose an interface called IObjectSafety. This interface contains functions that allow the container to request the control to enter safe for initialization or safe for scripting mode - as needed. The control can then respond with the degree of safety that it can actually support. This flexible approach to object safety is not supported by VB5 alone, but can be implemented in less than ten lines of code using SpyWorks 5.

Example 3: IShellLink

IShellLink is an interface that is exposed by the Windows 95 and Windows NT 4.0 interfaces to allow you to create, modify, and read file manager short cuts - files that typically have the extension .LNK. Like other non-automation interfaces, you can’t work directly with this interface using Visual Basic 5.0, but the SpyWorks dwEasy control exposes an object that lets you work directly with the IShellLink interface. SpyWorks 5 will continue to add support for interfaces of this kind. One of the most important of these included with SpyWorks 5 is the Desaware Enumeration object. This object supports the IEnumVariant interface, allowing you to add support for the Visual Basic For…Each command to any object, array or collection. This can be dramatically more efficient than the technique demonstrated in the VB5 documentation, where every custom collection contains a separate collection object. Now your collections can be based upon arrays or linked lists, still use enumerations and work with the For… Each command.

Conclusion

This article has detailed three different types of interfaces and the problems that they present. In one example it was necessary to override a portion of an existing implementation of an interface that is currently supported by Visual Basic - while leaving the rest of the implementation untouched. In another example it was necessary to add a completely new non-automation interface to an existing object. In the third example it was necessary to access an object using a non-automation interface. The SpyWorks 5 ActiveX Extension Library tackles all three of these problems, in some cases with solutions specific to standard interfaces, in others with generic solutions that you can extend in ways that we can’t even imagine - and you can expect the number and types of interfaces supported by SpyWorks to increase as time goes on.

Dynamic Export Technology

Consider the problem:

Visual Basic 5 does allow you to retrieve the address of a function in a standard module. But it does not let you export it. Exported functions are an essential part of some tasks in Windows including:

- Creating control panel applets
- Creating true NT Services (which use control panel applets)
- Creating export function libraries
- Creating ISAPI filters

How does one add support for exported functions to Visual Basic?

Well, you could perhaps reverse engineer the compiled Visual Basic DLL - but this presents a number of problems:

- What happens if Visual Basic changes their internal structure?
- How do you avoid compatibility problems in trying to make Visual Basic do something unsupported?
- How do you guarantee that your changes are automatically added to the DLL each time it is built?

All of these problems make this approach unacceptable. One of the reasons that our tools have been so successful over the long run is that, while they are low level tools, they are NOT based on hacks and odd reverse engineering. They are based on documented low level features that are built into Windows and on taking advantage of the fundamental design of features such as OLE. If anything, we apply an extremely paranoid approach to our components. In five years we have had only one feature designed out from under us - and that occurred during the transition from 16 to 32 bits - the feature continues to work well in the 16 bit environment for which it was designed.

So, from our point of view, anything that modifies your compiled component is a high risk approach. The only exception that we've ever made to this was in our VB3 VersionStamper product which embedded version information in compiled VB3 applications - but note that those parts of the executable file format that relate to version information are clearly and publicly documented - so it really was a safe thing to do!

Desaware's new SpyWorks 5 Dynamic Export Technology™ eliminates this problem by allowing you to export the functions immediately when your DLL loads. This is accomplished through an easy to use custom interface. The dwExport utility obtains a reference to your interface and calls functions which you implement. In these functions you define the names of the export functions, their ordinal values (every exported function can have an identifying number), and the standard module function that implements the export. Once the dwExport library has done it's work, whomever is using your DLL will directly access the standard module functions that you have exported.

We think that this is one of the most exciting new technologies introduced in SpyWorks 5 - a true Desaware exclusive.

VersionStamper 5: Using New Technology to Resolve File Conflicts

You have all heard the story. A customer calls, your application does not work anymore… VersionStamper has become the standard distribution toolkit for use by VB programmers to detect possible component version incompatibilities in their end-user's systems. VersionStamper 5 adds new support for the internet and corporate intranets. What does this mean? Now you can keep an updated list of file dependencies on your web or FTP site that your application can be programmed to use when verifying files on an end user's system. You can also keep a list of notification messages on the same site that VersionStamper will use to inform your users how to resolve problems on a case by case basis. When VersionStamper locates a problem or an updated file, it can notify the user, send email back to your company, or even download the correct components from your web or FTP site. VersionStamper is also useful for corporate IS departments which can now ensure that their clients really do have the right components on their system - before tying up the service desk for hours. For example, you can distribute a copy of a VersionStamper scanning application to each client machine. Each night, it can automatically scan the client machine based upon information supplied from the server. Notification regarding any conflicts or out-dated files located can be sent to an email address, identifying the client machine and the problem files, or you can choose to have the client machine automatically updated from files on the server. Adding new files or changing the file list is a simple matter of modifying the information on the server. VersionStamper also includes extensive sample code.

Dear Marian:

What is the difference between OLE Controls and ActiveX controls?

There is no difference. None - nada. Microsoft changed OLE to ActiveX by changing the name. A complete (and slightly irreverent) explanation of how and why this happened can be found in Dan's new book: Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed. Take a look at the article on page 3 of this issue.

Don't Buy This Book: PC Magazine Visual Basic Programmer's Guide to the Win32 API (VB 4 Edition)

If you currently own the “PC Magazine Visual Basic Programmer’s Guide to the Win32 API”, this important message is for you. As you know, this book is intended to provide Visual Basic programmers with the education and information that they need to become outstanding Windows programmers. All of the text and reference material is written from a Visual Basic programmer’s perspective.

With the appearance of VB5 it became necessary to update this book. The samples needed to be recompiled and tested (and occasionally updated) for VB5. They also needed to be tested, and sometimes updated, for Windows NT 4.0. The text needed to be updated for both.

But think about it: unlike most Visual Basic books, this book is really about Windows - and the core functionality of Windows hasn’t really changed that much. As a result, the updated version of the book is really substantially the same as the current one. There are a couple of new chapters (on the CD only, because at 1500 pages, the book itself is full), and some new articles. And of course, the changes just described. The book also has a new title: “Dan Appleman’s Visual Basic 5.0 Programmer’s Guide to the Win32 API” - for the most part because it really has nothing to do with PC Magazine. The book also has a new ISBN number, 1-56276-446-2, because it has a new title, and a new price of $59.99. This increase is due to the fact that when the book was originally priced Ziff-Davis (the publisher) had expected 1000 pages, and Dan (being Dan) delivered 1500, which did not at all please the publisher’s accountants who calculate production costs.

The greatest concern when negotiating to do the update was that readers might purchase the new edition thinking that they are getting a completely new book, and be disappointed when they discover that it is substantially the same. Desaware’s philosophy is to always provide excellent value along with excellent products. (Keep in mind, though, that Desaware offers a 20% discounts on these books - more for SpyWorks Pro subscribers.)

So here is what was finally agreed upon with the publisher for current owners of the API book: since the entire book contents are also present on the CD-ROM, Ziff-Davis agreed to allow Desaware to offer a CD-ROM only upgrade program at a substantial discount. The final details are still being worked out as this newsletter goes to press, but you can find details on our web site at www.desaware.com. Currently the expected the price of the upgrade is to be $25 plus a nominal shipping charge. Documentation requirements set by the publisher (for example: you may need to send in your current CD) will also have to be met. This offer is available exclusively through Desaware, so check our web site for details.

Dan Appleman's Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed

I consider ActiveX component and control development to be one of the most significant (and coolest) features of Visual Basic 5.0. It's the one that I'm most excited about, and as an author, I tend to want to write about things that I think are important (especially those things that I get excited about). But…

I hate manual rehashes - those seemingly thousands of books that basically regurgitate the Visual Basic documentation and claim to be the "Bible" for Visual Basic developers. The Visual Basic documentation on creating ActiveX components and controls is really pretty good - it's comprehensive, reasonably accurate, and there is a lot of it. The world does not need another "Bible" on ActiveX components and controls with VB5 - Microsoft has provided one. And I certainly did not want to write one.

I puzzled over this dilemma for a few weeks until this epiphany struck: What is a Bible, without a commentary? I resolved to write the definitive commentary to the Visual Basic documentation on the subject of ActiveX components and controls.

The VB5 documentation is very strong on tutorials, reference material, and demonstrating how to perform "typical" tasks. My book does not focus on these subjects at all. Instead, this book addresses the areas where VB5 documentation falls short: teaching underlying concepts, describing advanced and "creative" techniques, clarifying which features are truly important and which are "fluff", and explaining the underlying reasons why some things are done the way they are.

My hope is that this book will take the average intermediate level Visual Basic programmer and turn them into a true guru on ActiveX technology. The book does get quite advanced at times, but it builds up slowly, so anyone who has a basic understanding of Visual Basic's syntax will be able to follow the material with little trouble. Even programmers who are not knowledgeable about class modules or object oriented programming will be quickly brought up to speed on these subjects. Advanced programmers will find a great deal of useful information as well. Absolute beginners? Well, you might want to look elsewhere to learn Visual Basic first, though anyone (even non-programmer managers) will be able to understand the first two chapters, which lay out the philosophical and historical basis for ActiveX technology.

Published by Ziff-Davis Press, ISBN: 1-56276-510-8 - $49.99 (20% off when ordered directly from Desaware).

QT on the Registry

Microsoft Windows 95 has many features that ease the user's workload. The key to writing a program that uses all of these features is the Registry. The Registry is a hierarchically organized central database that replaces the flat WIN.INI file. Accessing the Registry allows you to add in any of those neat new Windows 95 features, transforming your program from a quirky, obviously Visual Basic executable into a professional program that fits into the new operating system.

Have you noticed that when you boot into Windows 95, any Windows 95 aware program that was left open when you last shut down starts up right where you left it? Your program can do this as well. When you want to start your program the next time Windows starts, simply add a value to the key: HKEY_CURRENT_USER\Software\ Microsoft\Windows\CurrentVersion\ RunOnce

The value name is the program identifier, and the value data is a string containing the command line to be run. For example, a program might add a value named "TestApplication" with the value "C:\VB4\ Programs\TestApp\Test Application.exe Document1.tst". You can add whatever command line arguments you need to restore the application to the same state it was left. The RunOnce key is cleared after all the programs listed in it have been launched, so there is no need for clean up.

The "RunOnce" key is great for continuing applications after the user has turned off the machine. But what about a program that needs to be run every time the user loads Windows, such as virus checkers? Simply enter the a value of the same format in the key HKEY_CURRENT_USER\Software\ Microsoft\Windows\CurrentVersion\Run

More useful places in the Registry are found in the StorageTools manual and in numerous Microsoft documents.

Product Descriptions

SpyWork Series:

SpyWorks is a toolkit that extends Visual Basic. Do virtually anything that you could do in other languages such as C++. SpyWorks comes in three different versions to meet your needs. Following are descriptions of these choices.

SpyWorks Professional™ Edition version 5 $249.00

SpyWorks Professional is the subscription based edition. If you want the latest technologies as we develop them - this edition is for you. You receive the 16 & 32 bit OCX (ActiveX), the 16 bit VBX edition, and three additional updates with application notes, new features and upgrades. SpyWorks 5 Professional now includes ActiveX Extensions, Dynamic Export Technology™, Desaware API Class Library, cross task controls, debugging tools, extensive source code and much more. SpyWorks also includes full VB5 source code for components that do in-process subclassing and hooks for intercepting messages, plus their big brothers for intercepting messages in other applications or throughout the entire system. Supports Win 3.1, Win 95 and up and Windows NT.

SpyWorks ™Standard Edition $129.00

SpyWorks Stardard is a subset of the Professional edition. It includes the core functionality of SpyWorks Professional but without the updates and VBX. SpyWorks 4 Standard is currently shipping, and is fully compatible with Visual Basic 5.0. SpyWorks 5 Standard, for Visual Basic 5.0, will be available Summer 1997.

SpyWorks™ version 2.1 $129.00

SpyWorks supports powerful Windows programming techniques that are not part of Visual Basic including subclassing, windows hooks, and callbacks. Provides virtual forms, rollups and many debugging tools. VBX version for VB 3 and VB 4 (16 bit edition).

VersionStamper™ version 5 $149.00

VersionStamper helps eliminate incompatibility problems that frequently occur when distributing component based applications by embedding into your executable version information about all of the DLLs and custom controls used by that application. Supports full component verification and component updates via the internet or corporate intranets. Configure any application to verify its own environment based upon embedded information or your web or FTP site. Component incompatibilities can be detected and handled with a report to the user, email message to you, or automatic update. 16 & 32 bit OCX (ActiveX) plus 16 bit VBX. Supports VB3-VB5, Windows 3.x, 95 and NT.

StorageTools™ version 1 $129.00

The ultimate data storage and file manipulation toolkit for Visual Basic and other OLE clients. Easily work with OLE 2.0 structured storage files from within your application. Tap into the full power of the registration database. Create resource files, and more. 16 & 32 bit ActiveX. Supports Windows 95, NT, VB4 and 5. Winner of the 1995 StarTech Award and 1995 VB Tech Journal Techie Award.

ActiveX Gallimaufry™ version 1.0 $79

A collection of ActiveX controls with full Visual Basic source code. Includes a hex editor, banner contol and spiral art control and much more!

Custom Control Factory™ version 4.0 $48.00

Create custom button style controls for Visual Basic & other full OLE Control clients.

Developing ActiveX Components with Visual Basic 5.0: A Guide to the Perplexed

A commentary and clarification of new Visual Basic 5 technologies written by Dan Appleman. Priced at $31.97.

Visual Basic 5.0 Programmer's Guide to the Win32 API/ The Visual Basic Programmer's Guide to the Windows API

Two best selling books for taking advantage of the Windows API. Written by Daniel Appleman, the president of Desaware. Both the 32 and 16 bit versions are available directly from us at a 20% discount, priced at $47.99 and $27.96 respectively.

The Win32 API includes an updated full text searchable CD Rom with additional chapters not found in the book, plus other technical data. At over 1500 pages, it has become the "must have" reference tool for VB developers. Available soon: the VB5 update on CD Rom!

SpyWorks 5 Schedule

SpyWorks is available in three different editions. The most popular of these is the Professional edition which is a subscription based product that includes three updates. We took this approach in order to balance the fact that most of our customers need to deploy new technology as quickly as possible. The subscription cycle allows us to ship the latest features and additional sample code as soon as it is available.

The 5.0 Professional update is scheduled to ship within a few weeks of the release of Visual Basic 5. This update will include all of the technologies described in this newsletter including the new ActiveX extensions, Dynamic Export Technology™ and dwSpyVB.dll Windows utilities. But it will not yet include all of the samples that we would like, or support all of the interfaces that are planned. It also lacks the final printed manual for SpyWorks 5 (though preliminary Help file documentation is, of course, included).

We expect to ship the 5.1 drop of SpyWorks during the summer. It will contain the final version 5 printed documentation, additional sample code, and new features based upon what we learn during the preceding months. At that time we will also ship SpyWorks 5 Standard edition.

SpyWorks Standard edition is a subset of the Professional edition. It does not include the SpyWorks source, and the various extension libraries lack some of the features of the Professional edition. (A complete list of differences can be found on our web site at www.desaware.com.)

SpyWorks 2.1, which is the VBX based edition for use with 16 bit editions of Visual Basic is also available and will continue to be supported.

By the way, if you are a current SpyWorks Professional customer whose subscription is expiring with the current drop, now is the time to renew - and it’s only $120 for the next three updates!

The Desaware Visual Basic Bulletin

Volume 3 Issue 2

Introduction

Those of you who have been reading our newsletter regularly are familiar with our newsletter philosophy - rather than drown you in glossy marketing material, we like to provide real value in the form of technical content. I'm pleased to say that this issue is one of the most technical issues that we've done. From subtle API debugging techniques, to intertask data transfer under Win32 - I think you'll find this issue to be more than worth your while. And with the release of my new book the PC Magazine Visual Basic Programmer's Guide to the Win32 API to the announcement of our new Internet web site: http://www.desaware.com - I hope you'll continue to find Desaware a solid source for advanced Visual Basic development products and information.

—Daniel Appleman

Inside...

Product Information:

(Note: These are the original product descriptions as they appeared in the newsletter. For more detailed information, refer to the Desaware online catalog)

SpyWorks 4.0 for 16 and 32 bit OLE Containers
StorageTools Version 1.0
VersionStamper version 4.0
The Visual Basic Programmer's Guide to the Win32 API
SpyNotes #2 The Common Dialog Toolkit
Classics from Desaware - CCF-Cursors and The Custom Control Factory
PC Magazine's Visual Basic Programmer's Guide to the Windows API

Jump to Newsletter #6
Jump to Newsletter #4
Jump to Newsletter #3
Jump to Newsletter #2
Jump to Newsletter #1

Goodbye GetModuleUsage! Hello what?

How to Find Out When a Shelled Application is Closed Under Win32

One of the most common Win32 questions that we've dealt with recently concerns the question of determining when a shelled application has closed under Win32. Under 16 bit operating systems, you may recall that the Shell function returns an instance handle (or module handle) that uniquely identifies the newly launched application. You could then use the GetModuleUsage API function to watch the module count for the application in a DoEvents loop - when the module count goes back to zero, it indicates that the application has closed.

If you have tried this under Win32 you may have noticed a few minor problems starting with the fact that the GetModuleUsage function does not even exist under Win32!

The reason it doesn't exist has to do with the underlying architecture of Win32 - an instance handle is fundamentally a reference to a block of memory, and under Win32 memory is no longer shared among applications. This means that an instance handle cannot uniquely identify a running task. Even if GetModuleUsage still existed, it would have no meaning because it would only be able to obtain the module count for its own instance. It couldn't possibly work on a module handle from a different application.

Ok, this is all very interesting, but knowing why GetModuleUsage won't work doesn't help solve the problem. How do you detect that an application has terminated under Win32?

The secret is to look at what methods Win32 does provide to uniquely identify a task in the system (by the way, tasks will henceforth be referred to by their Win32 name: processes).

Every Win32 process has a unique identifier called a process ID. This process ID is, in fact, the value returned by a successful Visual Basic "Shell" function call. But the process ID by itself is not very useful. In order to work with the process you need to obtain a handle to a process. A process handle is a 32 bit number that refers to the specific process. A process handle is obtained using the OpenProcess function and is valid only within the application that called the OpenProcess function. The code used to obtain a process handle is shown below. Just place a command button on a form and set its Name property to cmdWatch. You'll also need a program to launch (in this case we use project1.exe):

```
Private Sub cmdWatch_Click()
   Dim ProcessId&
   Dim hProcess&
   Dim ExitCode&

   ProcessId = Shell("project1.exe", vbNormalFocus)
   hProcess = OpenProcess(PROCESS_QUERY_ INFORMATION, False, ProcessId)
   cmdWatch.Enabled = False
   Do
      Call GetExitCodeProcess(hProcess, ExitCode)
      DoEvents
   Loop While (ExitCode = STILL_ACTIVE)
   cmdWatch.Enabled = True
   Call CloseHandle(hProcess)
```

```
End Sub
```

The PROCESS_QUERY_INFORMATION flag in the OpenProcess command tells Windows that you wish permission to query the process for its current status - there are many other possible permissions that you may request. We disable the command button to prevent reentrancy problems, then enter a DoEvents loop. This loop repeatedly queries the process to obtain an exit code. As long as it's STILL_ACTIVE, the process is still running. Finally, we re-enable the command button and close the process handle. Don't forget to close the process handle - until it is closed, the other application will not completely unload (even though it will be closed from the user's point of view, not all of its internal memory structures will be freed).

This code closely resembles the traditional DoEvents loop using GetModuleUsage, but it's still not an optimal approach. It burns up a great many CPU cycles with repeated calls to GetExitCodeProcess and DoEvents - cycles that are wasted. Have you ever wondered if there might be a more efficient way? Under Win32 there is a solution to this problem that is incredibly efficient. You can define a function in a form that will be called when the other application ends - until then your application can simply return from the function and continue to run. No DoEvents loop. No wasted CPU time.

This technique, which combines Win32 synchronization functions with OLE automation techniques, can be found in the Launch.vbp example on pages 1119-1135 of the Visual Basic Programmer's Guide to the Win32 API (along with a much more thorough discussion of instance handles, process identifiers, process handles and Win32 objects in general).

Windows API for Beginners: Using the Desaware Class Library

The Desaware API Class Library is a neatly organized collection of Visual Basic classes which makes it easier to access functions available to every program in Windows 3.1, Windows 95 and Windows NT. These functions are called the Windows API (Advanced Programming Interface). The Windows API functions have similarities to some of the functions which are available within Visual Basic. But sometimes you need to do more, do it faster, or do it more directly. That is when you turn to the Windows API functions. The best thing about these library files is that you can see how the functions are implemented because they are provided as Visual Basic class modules with full source code. What follows is a very simple example of how you can use a few of the hundreds of Windows API functions using the Class Library.

Each module in the Desaware Class Library encapsulates a certain Windows object or data type. Within the modules are functions for manipulating the object or data type. To create an instance of a class, use this format:

```
Dim InstanceName As New DwClassname
```

For example:

```
Dim FirstRectangle As New DwRECT
Dim FormWindow As New DwWindow
```

The example above allocates a space for a rectangle and a window using the DwRECT and DwWindow class files which are part of the Class Library. Now that the objects are allocated you can easily access the functions belonging to the class as long as you use the correct parameters. To find the correct parameters we suggest reviewing the Visual Basic Programmer's Guide to the Win32 API or actually examining the class files to learn which properties are needed to make the function work. Several of the function parameters have been slightly altered to make them easier to use. Let's review some of the functions available:

```
Public Sub MoveWindow(ByVal x As Long, ByVal y As Long, ByVal nWidth As
Long, ByVal nHeight As Long, bRepaint As Boolean)
' Code is here...
End Sub

Public Sub SetRect(ByVal x1 As Long, ByVal y1 As Long, ByVal x2 As Long,
ByVal y2 As Long)
' Code is here...
End Sub
```

You can get a general idea of what the function can do by looking it up, then all that is needed is to correctly specify the variables to get the function to work. In the case of MoveWindow, x and y refer to the top and left coordinates where the window is to be placed, nWidth and nHeight are the width and height of the window and bRepaint is True or False depending on whether the window should or should not repaint. In SetRect, x1, y1 represent the left and top coordinates of a rectangle and x2, y2 represent the right and bottom dimensions respectively. Here is an example of how to call these methods:

```
FirstRectangle.SetRect 10, 10, 20, 20
' sets the FirstRectangle rectangle
TestWindow.MoveWindow 10, 15, 400, 400, false
' moves the TestWindow based on the values inputted
```

Putting it all together is a snap. In all cases you need to make sure that the object is initialized using Dim or Set statements and you need to make sure that the proper files are added into your project. To ensure that you have the proper files, load one class at a time and run it. Each time the class is missing a file it will give an error message and highlight a line. On that line you will see a class object with the letters Dw at the beginning. Just keep adding those files to the project until you no longer receive error messages. Another technique to use to add files is to utilize Desaware's new Class Loader which will be available shortly.

For the examples in this article, you will need to add the following files:
dwwindow.cls, apigid32.bas, dwconst.bas, dwerrors.bas and dwtypes.bas

Here is the complete example of how you would initialize and use an instance of the dwWindow class:

```
Dim FormWindow As New DwWindow
' creates an empty window object
FormWindow.hwnd = Form1.hwnd
' Takes the handle of the Form1 window and puts it into the empty FormWindow
FormWindow.MoveWindow 10, 10, 200, 200, False
' Moves the form window to the values given
' Since the window FormWindow refers to Form1 this class method will move
Form1
' to the location specified
```

Here is the full dwRECT example:

```
Dim FirstRectangle As New DwRECT
' create an empty rectangle object
FirstRectangle.SetRect 10, 10, 25, 25
' Sets the Rectangle dimensions to the values given
Debug.Print "Rectangle Dimensions :" & FirstRectangle.Top,
FirstRectangle.Left, FirstRectangle.Bottom, FirstRectangle.Right
' Prints in the Debug window the position of the upper left and lower right
corners
' which make up the dimensions of the rectangle
```

The line

```
FormWindow.hwnd = Form1.hwnd
```

is crucial to the window function working because it sets the object FormWindow. Many times in using the Class Library you will need to use a line similar to this to set an object. The subroutine SetRect sets the rectangle. The MoveWindow operation will fail if the object is not set. Another problem you might encounter is an error message noting that a file is missing. To alleviate this make sure that the

apigidxx.dlls are in the same directory as the apigidxx.bas files.

The Class Library may look a little bit complicated but it’s very simple and practical. We suggest you attempt a few examples with different functions and try to get them to work. Once you know how to implement a few Class Library functions you will have hundreds of powerful functions at your finger tips.

New Features In StorageTools

Every registered user of StorageTools has received an update disk with the latest version of the StorageTools controls and the Resource compiler.

The StorageTools controls now have a number of features that have been requested by users. First, the speed of reading and writing arrays and user defined types has been improved dramatically. You will especially notice the difference in conditions where the drive cache is small, such as when accessing a file across a network. The Storage control can read and write pictures from the Picture property of forms, picture boxes, image boxes and other controls. It can now read and write blocks from a given address in memory, as well. This is especially useful for those of you who are using large blocks of memory in your 16 bit applications, or using data that is larger then the limits of Visual Basic arrays and strings. The 16 bit version of the Registry control can now access the full 32 bit registry when used in Windows 95 and Windows NT. This can help reduce the difference between the 16 and 32 bit versions of your product, and give your 16 bit applications amazing new capabilities that were impossible before.

We are constantly learning new things and improving our products. But we can’t send you the latest software if we don’t know where you are. If you have not yet registered your copy of StorageTools (or any Desaware product), do it right now!

Current Affairs

Desaware is currently scheduled to exhibit at VBits Orlando - Sept. 25th & 26th. Please stop by and say hello!

New Releases:

SpyWorks Professional Update: May

Includes the new 16 bit OCX, API Class Library plus updated 32 bit controls and additional software samples to all registered SpyWorks Professional subscribers.

SpyWorks Standard: May

16 & 32 bit OCX (only) version of SpyWorks.

Resource Compiler: June

Includes tools that allows you to easily embed bitmap, icon and string resources into your VB4 executables. Check out Dear Marian for more details.

SpyWorks Professional Update: Fall 1996

API Class Loader and other features as suggested by our customers.

Dear Marian...

When will Desaware be shipping ActiveX controls?

Actually we already are! You see, the only thing that you need to do to convert an OLE control into an ActiveX control is to call it an ActiveX control. All Microsoft did was change the name! Why did they do this? Well, you might want to direct that question directly to Microsoft. (We also sometimes find them hard to understand.)

What is the Desaware Resource Compiler and when will it be available?

The Desaware Resource Compiler is a tool that allows you to easily embed bitmap, icon and string resources into your VB4 executables (16 and 32 bits). The Resource Compiler runs as a standalone program or a VB add-in.

It includes complete VB source code and a technical description of how it works (portions of which have been serialized in Pinnacle Publications VB Developer newsletter). The Resource Compiler shipped late May and early in June to all registered users of StorageTools (which now includes the Resource Compiler as part of the package).

When will the next SpyWorks Pro Update be shipped?

We shipped the first SpyWorks Pro update in late April. The first update included the 16 bit OLE controls, 32 bit utilities programs, and the Desaware API Class library. The second update is schedule for Fall.

Who writes the articles for the Bulletin?

All of the articles are written by members of Desaware's technical staff. Our president, Daniel Appleman, writes some of the technical articles and, of course, those editorials that he signs personally. Others are written by Franky Wong and Stjepan Pejic. In this issue we also have a beginner's article by Rami Nagel.

SUPER SPY: Find the Bug

Using the LB_ITEMFROMPOINT Message

This article is one of a recurring series on technical questions and solutions.

Every now and then we receive a request for help from a programmer who is having trouble making an API technique work. In this case, the programmer wanted to use the LB_ITEMFROMPOINT message under Windows 95 to determine which item was clicked during a MouseDown event on a list box. Here's the code that we received - see if you can find the problems (the description of the LB_ITEMFROMPOINT message can be found on page 1387 of the VB Programmer's Guide to the Win32 API):

```
DefInt I
DefLng L

Private Declare Function SendMessage Lib "user32" Alias "SendMessageA" _
  (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, lParam As
Any) As Long
Private Declare Function SendMessageBynum Lib "user32" Alias "SendMessageA"
_
  (ByVal hwnd As Long, ByVal wMsg As Long, ByVal wParam As Long, ByVal
lParam As Long) As Long

Private Const LB_ITEMFROMPOINT = &H1A9

Private Sub Form_Load()
   Dim i
   List.Clear
   For i = 0 To 10
      List.AddItem
      list - item & i
   Next i
End Sub

Private Sub List_MouseDown(Button As Integer, Shift As Integer, X As Single,
Y As Single)
   Dim lpRetVal
   lpRetVal = SendMessage(List.hwnd, LB_ITEMFROMPOINT, 0, Y)
   Debug.Print lpRetVal
End Sub
```

How many problems did you find?

1 - Give yourself a point.

2 - You're an experienced API programmer.

3 - Congratulations!! You're an API guru.

Still stuck? Before you give up and look at the answers , try taking another stab at it by applying the ten commandments for safe API programming found on pages 127- 130 of the VB Programmer's Guide to

the Win32 API.

Answers to the SUPER SPY Exercise

Here are the answers:

1. This programmer forgot the first commandment of API programming (page 127): "Remember ByVal and keep it wholly." Not to mention commandment #5: "Thou shalt not use 'As Any', for it is evil." SendMessage has lParam defined 'As Any' - in this case the parameter needs to be a long value and passed ByVal. The solution: Use SendMessageByNum.

2. Read the documentation - then apply commandment IX: "Thou shalt check parameter and return values." The Y parameter needs to be placed in the high order word, not the low order word. In this case, the easiest way to shift the values is to multiply Y by &H10000. You can define the entire point using Y * &H10000 + X.

3. API functions use pixel coordinate systems. X and Y here are in twips. The list box control does not have a ScaleMode property which can be set to pixels, so you need to do the conversion manually using the TwipsPerPixelX and TwipsPerPixelY properties of the VB Screen object.

Here then, is the correct code.

```
lpRetVal = SendMessageBynum(List.hwnd, LB_ITEMFROMPOINT, 0, Y /
Screen.TwipsPerPixelY * &H10000 + X / Screen.TwipsPerPixelX)
```

Keep in mind that the high word of lpRetVal may be set to non-zero in some cases if the point is outside of the visible area of the control.

Add File Manager (or Explorer) Drag-Drop Support to Your Visual Basic Application!

File Drag-And-Drop is becoming a popular method for passing file information between applications. One typical use of file drag-and-drop is to use File Manager or Explorer to drag files into an application that can accept dropped files. This will pass the file names and paths of all the files dropped to the receiving application. In order for a window to receive dropped files, it has to have a special style flag set. Setting the WS_EX_ACCEPTFILES style flag for a particular window allows that window to accept dropped files. You can call the DragAcceptFiles API function (from the shell library) to set this flag. When a file is dragged outside of File Manager or Explorer, the mouse cursor is constantly monitoring the window it overlays, testing to see whether the window's WS_EX_ACCEPTFILES style bit is set. Once a file is dropped onto a window that accepts dropped files, a WM_DROPFILES message is posted to that window. Upon receiving this message, you may call the DragQueryFile API function to retrieve the names and paths of the dropped files.

It's easy with dweasy

Unfortunately, Visual Basic does not support this feature since it requires subclassing the WM_DROPFILES message. You can use SpyWorks' subclass control to do this but the new drag-drop feature in dweasy makes this even easier. Dweasy has a new property call DetectDropped. When this property is set to True, it sets the WS_EX_ACCEPTFILES style for dweasy's container and all the container's child windows. Dweasy will then subclass these windows for the WM_DROPFILES message. Setting another new dweasy property, FileDropHwnd to a window handle will set the WS_EX_ACCEPTFILES style for that specified window and cause dweasy to subclass it for the WM_DROPFILES message. When files are dropped to a window subclassed by dweasy, the dweasy's FilesDropped event is triggered. Inside this event, you can access dweasy's DroppedFile array to retrieve the paths and names of the files that were dropped.

How easy is it? Here's a sample program that will display a list of droppedfiles. Place a list box and a dweasy control on a form. Set the DetectDropped property for the dwEasy control to True. Add the following code to the FilesDropped event.

```
Private Sub SbcEasy1_FilesDropped(ByVal FileCount As Long)
   Dim cnt&
   For cnt = 0 To FileCount-1
      List1.AddItem SbcEasy1.DroppedFile(cnt)
   Next cnt
End Sub
```

That's all there is to it!

Passing Information Between 32 bit Applications

Passing information between applications was easy under Windows 3.1. You would simply create a global memory block with the GlobalAlloc API function GlobalAlloc, get the address of this block of memory with GlobalLock, and pass it to the other application. The first Desaware newsletter had an article about this subject, and sample projects, called COOP1 and COOP2, came with the newsletter's disk.

However, if you tried to use this method in your 32 bit applications, you probably found out that it no longer works. Windows NT and Windows 95 both have strong separation between applications. The separation means run-away programs can no longer damage other applications or the operating system with errant memory access. It also means that a valid memory address in one application may not be valid in another.

Distributed with SpyWorks 4 are two new sample programs called COOP1_32 and COOP2_32, which illustrate how two applications can access the same memory address (by using the dwXCopyDataFrom function in SpyWorks 4). However, this method requires that you also somehow inform the other application of your process ID and the size of the block of memory.

Another method for passing data between 32 bit applications can be found in the Visual Basic Programmer's Guide to the Win32 API. But even if you read all 1471 pages, you probably missed it. It is the WM_COPYDATA message, and it is designed to convert addresses between applications. The long parameter holds a pointer to a structure called a COPYDATASTRUCT defined as:

```
Type COPYDATASTRUCT
   dwData As Long
   cbData As Long
   lpData As Long
End Type
```

The lpData variable holds the address of the block to be shared between applications. The size of the memory, in bytes, is held in cbData. Any other information that the sending application needs to pass along can be placed in dwData. This copy is one-way only, and you should not write to the address you receive. Here is how a program might send this message:

```
Const WM_COPYDATA=&H4A
Dim cds As COPYDATASTRUCT
' Get address of the block and send it to the other
' application.

cds.lpData = dwGetAddressForObject(block)
cds.cbData = LenB(block)
' Note that we get the address of the first element in
'the structure to get the address of the structure.
lp = dwGetAddressForObject(cds.dwData)
ret = SendMessageBynum(OtherAppHwnd, WM_COPYDATA, Me.Hwnd, lp)
```

In the receiving application, set the Subclass control to intercept WM_COPYDATA messages.

You can extract the data like this:

```
Dim lpBlock As Long
Dim cds As COPYDATASTRUCT
Dim lpcds As Long ' pointer to the COPYDATASTRUCT
Dim Block() as Byte
' Of course, the block could be any contiguous area in memory.

' Get the address of the local COPYDATASTRUCT and
' copy the one pointed to by the lp parameter to
' the local one.
lpcds = dwGetAddressForObject(cds.dwData)
dwCopyDataBynum lp, lpcds, Len(cds)
' Now get the address of the local block, and copy
' the information at the address in the 'COPYDATASTRUCT to it. But first,
make sure
' the block is big enough hold the information.
ReDim Block(cds.cbData)
lpBlock = dwGetAddressForObject(Block(0))
dwCopyDataBynum cds.lpData, lpBlock, cds.cbData
```

If you use StorageTools, this can be used as a method for passing memory based storages from one program to another.

Writing a System Wide Macro Launcher

This article will step through the process of designing, coding, and debugging a macro launcher. This simple Visual Basic utility detects a set of hot keys and launches a command when they are detected. To keep things simple, this example sends a string of text to the active window when a hot key is detected and supports up to four distinct keys.

Initial Design

In designing this application, we allow the user to assign four hot keys to use. Each hot key will send text from a different text box. In order to detect hot keys in a 32 bit operating system, the application is best written as a 32 bit program. For this exercise, we assigned the Ctrl+F2, Ctrl+F3, Ctrl+F4, and Ctrl+F5 keys as the hot keys. We will use the Visual Basic SendKeys command to send the text from each of the designated text boxes. We use the Windows hook control from SpyWorks to detect the hot keys to trap. The hook control is set to monitor the entire system for the specified hot keys. When the hot key is pressed, an event in the SpyWorks hook control is triggered. The event code determines which hot key was pressed calls SendKeys with the appropriate text. Listing 1 shows the initial code:

```
Private Sub WinHook1_KeyDownHook(keycode As Long, ByVal shiftstate As
Integer, _
Repetitions As Integer, keystate As Integer, ByVal ProcessID As Long,
discard As Integer)
   Select Case keycode
      Case macrokey1:
         SendKeys Text1.Text
      Case macrokey2:
         SendKeys Text2.Text
      Case macrokey3:
         SendKeys Text3.Text
      Case macrokey4:
         SendKeys Text4.Text
   End Select
   keycode = 0
   discard = True
End Sub
```

Pitfalls

At first glance, this seems like a straight forward design for a simple application. When the coding was completed and the application was tested, things did not turn out the way we thought it would. For testing purposes, I opened up Notepad and hit the hot key while Notepad was the active window. The first time I tried this, everything worked perfectly. But several surprises unfolded during the course of testing. The first was that if I hit the Ctrl key and then the F2 key then released the F2 key but paused a while before releasing the Ctrl key, the expected text from the Text1 text box did not arrive. At first, I thought that because I was doing the SendKeys command during the KeyDown event, Windows got confused because a SendKeys command occurred in the middle of a key press. I tried this again after moving the code to the hook control's KeyUp event but got the same results. What is going on here? For testing purp oses, I created a utility program that does a SendKeys command during a timer event. Using this sample program, I proceeded to test SendKeys on Notepad. This program sent the correct text to Notepad, but if I held the Ctrl key down during the SendKeys I saw the same garbage text as I

saw with the macro launcher. I concluded that Keys sent via SendKeys are combined with the current state of the Shift, Ctrl, and Alt keys.

Back to the Drawing Board

Well, how do we get around this problem? One thing we can do is to restrict the hot keys from using Ctrl, Shift, and Alt keys, but this is probably not practical as those keys are often used as hot keys. Another solution is to wait for the Ctrl key to be released before calling SendKeys. I added the Ctrl key to the list of keys that the hook control will detect, then added a timer control and moved the SendKeys command to the timer's timer event. Finally, I added a global variable to hold the hot key code when a hot key was detected. I changed the code to enable the timer control when the Ctrl key was detected. This solution seemed to handle the problem, but I ran into one last obstacle. What if the SendKeys command sent a string of text which contains the hot key? This naturally resulted in an infinite loop. Fortunately, it is unlikely that you would include a hot key string within the SendKeys command. A simple solution might be to add code to disable the hook control before the SendKeys command, then enable it after the SendKeys command is complete. This approach was not sufficient to solve the problem - an infinite loop still occurred. If you recall the format of the SendKeys command in Listing 1, you will notice that only one parameter is passed.The second parameter (the "wait" parameter) defaults to False. This indicates that SendKeys will return process control immediately and will send the key sequences at a later time. This means that we would have enabled our hook control to detect hot keys again before the text has been completely sent by SendKeys. To fix this problem, simply set SendKey's wait parameter to True.

Final notes

Calling the SendKeys command with wait set to True slows down SendKeys considerably when sending keys to other processes. I experienced a noticeable delay when sending long strings. SendKeys seems to break the string up into smaller strings and sends them one at a time, resulting in a delay between each sub string. If you do not anticipate SendKeys sending the same key sequences as your assigned hot key, you are better off leaving the wait parameter set to False. The final sample project also includes an option to use the clipboard to transfer the text to the active Window via a clipboard "copy" and "paste." The complete project can be found on our web site at www.desaware.com.

The Desaware Visual Basic Bulletin

Volume 3 Issue 1

Introduction

Welcome to the fourth issue of our newsletter. The first thing you'll notice is that the wait for this issue was much shorter than last time. We are trying to come out with 3 or 4 issues a year. One thing that we are considering is whether or not to switch to online distribution - via Email or a World Wide Web site. What do you think? (send comments to 74431,3534 on Compuserve or 74431.3534@compuserve.com) or drop us a postcard.

Among the technical articles you'll find in this issue include an intriguing way to use OLE structured storage to create self-persistant classes (classes that save and restore themselves - a great way to design a reliable document format). Also a way to place your logo (or any other picture) in the client area of an MDI form.

And on the product front, this is the first announcement of the new Desaware API Class Library - the easiest way to use the Win32 API (and the best way to learn it, since we provide all the source code!). And yes, at long last, the Visual Basic Programmer's Guide to the Win32 API is at the printers. It's been a lot of work, but at long last all of our 32 bit OLE control products are shipping and available - and we're even offering a developer's suite that can save you quite a bit on what I think are some of the coolest tools in the business. So dive in and enjoy!

Daniel Appleman

Inside…

Product Information:

(Note: These are the original product descriptions as they appeared in the newsletter. For more detailed information, refer to the Desaware online catalog)

SpyWorks 4.0 for 16 and 32 bit OLE Containers
StorageTools Version 1.0
VersionStamper version 4.0
The Visual Basic Programmer's Guide to the Win32 API
SpyNotes #2 The Common Dialog Toolkit
Classics from Desaware - CCF-Cursors and The Custom Control Factory
PC Magazine's Visual Basic Programmer's Guide to the Windows API

Jump to Newsletter #6
Jump to Newsletter #5
Jump to Newsletter #3
Jump to Newsletter #2
Jump to Newsletter #1

Creating Custom MDI Form Backgrounds

Many programmers take advantage of the Multiple Document Interface (MDI) standard as the basis for their application. As nice as MDI forms are, if you use them you are stuck with the bland, plain solid background color offered by Visual Basic. The other day I received a message from a programmer who said that if only we offered a control that would let him put a custom logo or picture on the MDI form background, he would be the first to buy it.

I had to disappoint him, because as you will soon see - SpyWorks has supported this capability for ages, and now supports it under 32 bit VB4 as well.

After several false starts (which are described in chapter 17 in the Visual Basic Programmer's Guide to the Win32 API), I came up with the following code that works very nicely.

Add a picture control to an MDI form at the top or bottom if you don't have one already - you'll need it to hold the dwsbc32.ocx control. Don't worry if you don't want the picture control to appear in your application - you can simply turn the Visible property for the control to False to hide it. Load a bitmap into the picture control (or a second child picture control). Drop a dwsbc32.ocx control on the first picture control and use the Messages property to intercept the WM_ERASEBKGND message. We're going to completely take over the background drawing for the MDI form. Set the Type property to Pre-Default. The declarations you'll need are shown in Listing 1. In this example they are declared as private declarations for use in the MDI form code. You can also declare them as public and place them in a module.

In the form load event for the MDI form you'll need the following code:

```
SubClass1.HwndParam = GetWindow(hwnd, GW_CHILD)
Form1.Show
```

What's going on here? Every MDI form actually consists of two windows. The background area of the form is a separate window from the MDIform - it is a window with the class MDIClient. The hWnd property for the form returns the form window handle, but we need to subclass the MDIClient window - thus we use the GetWindow function to retrieve the handle to the child window.

The real work is done by the SubClass1_WndMessage event. The device context for drawing is provided as the wp parameter for the function. Normally, this device context has a clipping region set so that only those areas that need to be drawn are accessable by the device context. In this case, we must clear the clipping region because we need to override the normal scrolling operation provided by the form. (Try commenting out the SelectClipRgn function and see what happens when you scroll the MDI form.)

We create a temporary compatible device context and select into it the handle of the bitmap that we want to display. The FillRect function draws the main background area (outside of the bitmap area). The BitBlt function copies the bitmap into the desired location on the form.

Note how the bitmap and compatible device context are cleaned up after they are used. The retval event parameter is set to True to notify Windows that the background has been erased. The nodef event parameter is set to True to prevent the default WM_ERASEBKGND message processing from taking place.

Figure 1 shows the results. The sample code can be found on the CD-ROM that accompamies the VB Programmer's Guide to the Win32 API.

Listing 1

```
Option Explicit
GetWindow() Constants
Private Const GW_CHILD = 5
Private Declare Function GetWindow Lib “user32" (ByVal hwnd As Long, ByVal wCmd As Long) As Long
Private Declare Function CreateCompatibleDC Lib “gdi32" (ByVal hdc As Long) As Long
Private Declare Function GetDC Lib “user32" (ByVal hwnd As Long) As Long
Private Declare Function ReleaseDC Lib “user32" (ByVal hwnd As Long, ByVal hdc As Long) As Long
Private Declare Function DeleteDC Lib “gdi32" (ByVal hdc As Long) As Long
Private Declare Function SelectObject Lib “gdi32" (ByVal hdc As Long, ByVal hObject As Long) As Long
Private Declare Function BitBlt Lib “gdi32"  (ByVal hDestDC As Long, ByVal x As Long, _
ByVal y As Long, ByVal nWidth As Long, ByVal nHeight As Long, ByVal hSrcDC As Long, _
ByVal xSrc As Long, ByVal ySrc As Long, ByVal dwRop As Long) As Long
Private Declare Function GetObjectAPI Lib “gdi32" Alias ”GetObjectA" (ByVal hObject As Long, ByVal nCount As Long, lpObject As Any) As Long
Private Declare Function GetClientRect Lib “user32" (ByVal hwnd As Long, lpRect As RECT) As Long
Private Declare Function DeleteObject Lib “gdi32" (ByVal hObject As Long) As Long
Private Declare Function FillRect Lib “user32" (ByVal hdc As Long, lpRect As RECT, ByVal hBrush As Long) As Long
Private Declare Function SelectClipRgn Lib “gdi32" (ByVal hdc As Long, ByVal hRgn As Long) As Long
Private Declare Function ExcludeClipRect Lib “gdi32" (ByVal hdc As Long, ByVal X1 As Long, ByVal Y1 As Long, ByVal X2 As Long, ByVal Y2 As Long) As Long
Const COLOR_BACKGROUND = 1

Private Const

Private Type RECT
        Left As Long
        Top As Long
        Right As Long
        Bottom As Long
End Type
‘ Bitmap Header Definition
Private Type BITMAP
        bmType As Long
        bmWidth As Long
        bmHeight As Long
```

```
        bmWidthBytes As Long
        bmPlanes As Integer
        bmBitsPixel As Integer
        bmBits As Long
End Type
```

Listing 2

```
Private Sub SubClass1_WndMessage(hwnd As Long, msg As Long, wp As Long, lp
As Long, retval As Long, nodef As Integer)
    Dim tdc&
    Dim usedc&
    Dim oldbm&
    Dim bm As BITMAP
    Dim rc As RECT
    Dim offsx&, offsy&

    ‘ Get a DC to draw into
    usedc = wp
    ‘ Create a compatible DC to use
    tdc = CreateCompatibleDC(usedc)

    ‘ Gets the bitmap handle of the background bitmap
    oldbm = SelectObject(tdc, Picture1.Picture)
    Call GetObjectAPI(Picture1.Picture, Len(bm), bm)
    Call GetClientRect(hwnd, rc)
    ‘ Decide where to place the MDI client logo
    offsx = 20
    offsy = 20
        ‘ Set the clipping region to the entire window -
    ‘ necessary because the hDC provided has a clipping
    ‘ region set.
    Call SelectClipRgn(usedc, 0)

    ‘ We exclude the bitmap area - this reduces flicker (try removing it)
    Call ExcludeClipRect(usedc, offsx, offsy, offsx + bm.bmWidth, offsy +
bm.bmHeight)
    Call FillRect(usedc, rc, COLOR_BACKGROUND)

    ‘ And restore the clip region before painting the bitmap
    Call SelectClipRgn(usedc, 0)

    Call BitBlt(usedc, offsx, offsy, bm.bmWidth, bm.bmHeight, tdc, 0, 0,
SRCCOPY)

    Call SelectObject(tdc, oldbm)
    Call DeleteDC(tdc)
    nodef = True
```

```
    retval = True
End Sub
```

SpyMem

Have you ever wondered how much memory and how many resources are being used by your application? Have you ever wanted to find resource leaks - in which your program (or others) allocate resources but fail to free them? Have you ever wondered what DLLs or VBXs your application has loaded? SpyMem has the unique capability of being able to take a snapshot of your current system state, be it global memory, resources or loaded modules. This snapshot is taken by displaying the list that your are interested in and clicking on the "Set Reference" button in the SpyMem main window.

You can then perform any operations, and later compare the system state with the saved reference. The comparison displays a list of items that are not present in the reference list. This feature can be applied in several ways. by taking a snapshot before you run your program and comparing the module list after wards you can obtain a list of new middles that were loaded by your applications, make it possible to detect DLLs and VBXs that were newly loaded by your application. You can take a snapshot of system resources before running your program, then run the program, close it, and perform a comparison. This will give you a list of resources that were created during the load an exit process that were not freed. These may be a caused by a resource leak. SpyMem's ability to obtain detailed information about resource heap objects can be very helpful in identifying the exact resource that is not being released (for example: not only will SpyMem identify an object as a bitmap, it lets you view the bitmap. A pen is not just identified by type, but by color as well).

SpyMem has been improved for the current version of 2.1 and 4.0 Professional. It includes the ability to display detailed statistics about the GDI resource heap such as the percentage of the allocated heap used, number of objects, number of objects of each type, etc. Filtering will make it possible to list memory blocks for a single application, allowing you to focus on the task at hand. A new footprint analysis tool will automatically take a snapshot of global memory and the GDI and User heaps, launch a specified application, and display a summary of the memory and resources used in loading the application, allowing you to determine the demand of an application on the system.

Current Affairs

Things They Are A Changin'

We would like to announce that as of January 1, 1996, we have changed our name to:

Desaware, Incorporated
1100 East Hamilton Avenue, Suite 4
Campbell, CA 95008

Our phone and fax numbers remain unchanged.

When you update your records, don't forget the address change which we announced in our last newsletter. Thanks.

Come cruise with us!

Check out our new home page on the web. It includes the latest product information and demos. Our address is:
HTTP://www.desaware.com

You are also welcome to visit our forum section on Compuserve, "Go Desaware"

On the road again

We will be exhibiting at the following venues in the near future. Stop by and say howdy!

San Francisco February 12-14

StorageTools Garners a Star Tech and a Techie Award

We would like to thank the editors of Windows Tech Journal for awarding StorageTools the Star Tech Award for 1995. This award was given to just *four* products in recognition of outstanding innovation, significance and usefulness as a software development tool for Windows programmers. The citation appears on page 27 of the December 1995 issue of Windows Tech Journal.

We are also proud to announce that StorageTools was one of *six* products awarded a Techie Award as one of the very best new products of the year for Visual Basic programmers. This from the editors of VB Tech Journal. The article appears in the December issue of VB Tech Journal on page 16.

Thanks again for both honors.

The Visual Basic Programmer's Guide to the Win32 API

The original "Visual Basic Programmer's Guide to the Windows API" has, thanks to your support, gradually become one of the standard references for Visual Basic programmers. It is also the unofficial "real" manual for Desaware's SpyWorks. The tie in between the two is not surprising, and it goes beyond the fact that I am both the founder of Desaware and author of the book. Both the book and SpyWorks have one primary goal: to empower Visual Basic programmers by giving them the information and tools to tap into the full power of Windows from their Visual Basic applications.

As this newsletter goes to press, the book is in the final stages of production and on its way to the printer. I realize that many people wish it were available earlier, with the arrival of Visual Basic 4.0 - but allow me to point out an often unspoken but obvious fact. *Any VB4 book that is available within a few days or weeks of Visual Basic 4.0 cannot possibly be based on the final released version of VB4!* It takes time to write a book, and typically takes at least a couple of months after the author has done his job for the book to get into print and become widely available. Even a multi-author work, which can be written quickly, cannot be available with the product and be 100% accurate. Visual Basic 4.0 underwent significant changes close to the release date. I decided, for better or worse, that the VB Programmer's Guide to the Win32 API would be based on the final VB4, Windows NT 3.51 and Windows 95 products - even though that meant that it wouldn't appear until several months after VB4 shipped.

That said, allow me to add that I am very excited by the results, and I think that you will be as well. The amount of explanatory text has increased substantially (I like to think that I've learned a thing or two about writing in the past few years). The number of examples has increased dramatically. There is a great deal of information on porting applications from 16 bit and writing applications that will run on both 16 and 32 bit platforms. There are sections that explore the internal workings of Visual Basic with regards to API calls. It is an almost complete rewrite of the 16 bit book, yet the overall structure remains the same. I don't have the final page count yet, but it's big - very big. Even though the emphasis was on making the core book better - there are a number of cool new features that I think you'll be very pleased with. Here's a short list:

CD-Rom with a full text searchable edition of the book: Let's face it - it can be hard to find something in a large book even if you know where things are. A text search can make an enormous difference.

The Desaware API Class Library. Yep, Desaware has come up with an API Class Library (or object library, if you will), that encapsulates large parts of the core API. SpyWorks Professional customers will gain the full benefit of this library (as they will receive the latest edition 3 times/year along with new advanced classes for subclassing and other functionality). But a large portion of the core API classes (dwSystem, dwWindow, dwHDC, dwDevice context etc.) are included with the book. Not only that - but they are included with full VB source code - so even if you prefer not to use a full class library, they serve as a source of examples of how to implement hundreds of functions.

The Desaware API Toolkit. A number of goodies here including: An API text viewer that works as a VB add-in, supports 32 and 16 bit conditional compilation, and includes full source code. An API database that fixes a number of errors from the win32api.txt file (well, hundreds of errors, actually - but who's counting?).

Other cool things. Some articles I've written in the past. Some demo controls. Some video clips intended to help explain tough concepts. And whatever else I thought you might enjoy.

And for those of you who were at VBits Florida and were surprised by the $59.95 price on the flyer - good news, I was succesful in convincing the publisher to lower the price to $49.99. Add to that the

20% discount that you can get by ordering the book directly from Desaware, and I think you'll find it one of the best values around.

So stay tuned and watch for it (or even pre-order) - it should be available very soon. And in the meanwhile for those of you who don't yet have the original Visual Basic Programmer's Guide to the Windows API and are wondering if you should get it - it's your call. If you don't need to do 16 bit API programming, wait for the new one. But the Win32 book does not cover Win16 at all - other than in the context of porting software or writing software that is compatible with both platforms - so if you are still doing 16 bit programming, that is the book you'll need even after the Win32 book is available. The 16 bit version of the book is currently available from Desaware at $27.96 plus shipping & handling.

Subclassing and Process Spaces under Win32

(*Portions of the following article are excerpted from the Visual Basic Programmer's Guide to the Win32 API..)*

You've probably heard that Windows 95 and Windows NT are considerably more stable than Windows 3.x. But what does this really mean? To a large degree it means that it is much more difficult for one application to crash another application or the entire system. Under Windows NT it is nearly impossible. Windows 95 is not quite as stable, but it is still better than 16 bit Windows.

How is this stability accomplished? One of the most important mechanisms has to do with the way Windows organizes memory. This is something that most Visual Basic programmers rarely need to think about, but it is both interesting, and important to know when you start working with 32 bit subclassing and hooks.

Imagine, for a moment, that you have a system with 16 megabytes of memory. The physical addresses of memory will range from address 0 through &H1000000. For now we'll ignore the issue of virtual memory, in which the hard disk is used to simulate additional memory. Purists will also note that this discussion will ignore the implications of the segmentation architecture used under Windows 3.x - an issue which would complicate the subject considerably and distract from the principles that are the focus of this article.

In this example assume that three programs are running, the Program Manager, File Manager and a user application. Under Windows 3.x, they might be organized in memory as shown in Figure 1. A couple of system dynamic link libraries are shown as well. Each of these modules needs to access memory in the normal course of running, and uses pointers internally to reference locations in memory. If the program is working correctly and has no bugs, it will never try to access memory outside of the memory space that is allocated to it (except under certain special conditions, as you will see). However, if something goes wrong or the program has a bug that gives one of these pointers an invalid value, it is possible for the program to write into the memory belonging to another application or even the operating system. This means that one application has the potential to corrupt memory on a systemwide basis.

Figure 1: Allocation under Windows 3.x

Figure 1 - Memory organization under Windows 3.x

This memory organization does have one advantage when it comes to performing subclassing. Since each application can easily access memory belonging to other applications, it is relatively easy for an application to hook into another application's internal data and substitute its own function address for the

window function of any window belonging to that application. It would not be accurate to say that this type of cross-task subclassing is simple - there are many other issues involved in doing it safely, but it certainly is easy compared to what comes next.

As you have seen, under Windows 3.x each application shares the same view of system memory and can access all available memory. Windows NT uses a much more advanced memory scheme in which every memory access by an application undergoes a translation step that converts it into a physical memory address. This fact has far reaching implications. For one thing, since each application's memory access has to be translated anyway, there is no reason why more than one application should not use the same addresses. This is, in fact, what Windows NT does - each application is given an address space of 4 gigabytes - that's &0 through &FFFFFFFF&. This does not mean that you need 4 gigabytes of memory or even disk space to use Windows NT. It does mean that each application can allocate blocks of memory anywhere in that address range and have the operating system assign those blocks to real memory. Figure 2 illustrates how two applications might view memory and how it might be translated into physical memory.

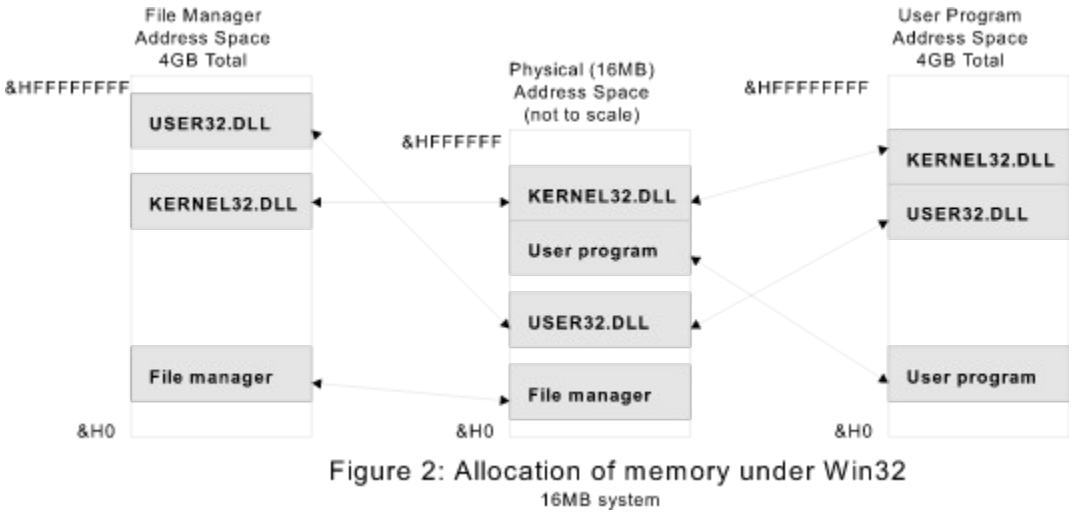

Figure 2: Allocation of memory under Win32
16MB system

Figure 2 - Memory Organization under Win32

The first thing you might notice is that there is no relationship between the location of data or code in the address space of the application and its location in physical memory. In fact, a dynamic link library that is shared by two applications might appear at completely different addresses as viewed by each application. Also note that the memory for one application does not even exist in the address space of the other application. This means that it is completely impossible for one application to accidentally write into physical memory assigned to another - and difficult to do intentionally (and only possible when explicitly requested through the operating system). It also means that an address that is valid in one application, is almost certainly not valid in another.

What about Subclassing?

Despite these obstacles, the SpyWorks 4.0 subclassing control dwsbc32.ocx, has no trouble subclassing windows that belong to other applications. Well, to be fair - it does have trouble, but that trouble is almost completely hidden from the Visual Basic programmer.

The secret to how this is accomplished lies in the dwspy32.dll module - the 32 bit subclassing engine. You see, it really is impossible for one application to directly subclass another. But it is possible for a dynamic link library to be loaded by more than one application at once. As a first step toward any cross task operation, dwspy32.dll maps itself into the other application's address space. This means that the other application becomes able to execute code in dwspy32, just as each application is able to execute operating system code. dwspy32.dll is able to keep track of which applications are mapped and has

the ability to communicate from one address space to another. When you try to subclass another application's window using dwsbc32.ocx, the control tells dwspy32.dll which window it wants to subclass. dwspy32.dll checks to see if the window is already subclassed - if so, it just notes that your application wants to intercept messages for that window as well. If it is not already subclassed, dwspy32.dll maps itself into the other application's address space and subclasses it in that address space. Any time a message comes in, dwspy32.dll notifies your application and triggers the appropriate event.

A similar mechanism comes into play for windows hooks and keyboard hooks - any time you try to work with information outside of your own application or on a systemwide basis, dwspy32.dll takes over.

Dealing with Pointers

There is still one catch - once you subclass a window belonging to another process, some of the information you retrieve will consist of memory addresses in the other application's address space. For example: the WM_GETTEXT message returns a pointer to a string containing a window caption. Due to the memory translation scheme, this pointer is useless in your application - it only has meaning in the other program's address space. For this reason dwspy32.dll provides a set of cross task memory functions that are designed to make it easy to safely read and write memory belonging to other applications. Safely in this context is relative - if you are careful, you should be fine, but this ability to access another program's memory space has the potential of bringing back the problems that are inherent in Windows 3.x - the ability of one program to crash another or the entire system. The SpyWorks tools always strive to let you keep your level of invasiveness to a minimum.

What about Windows 95?

Why the focus so far on Windows NT? Because NT provides the highest level of security available. Under Windows 95, the separation between applications is not quite as extreme. For example: the upper two gigabytes of memory are actually shared among all applications. This area in memory contains dynamic link libraries and operating system modules that are mapped into the same addresses for all applications. Application modules, which are placed in the lower two gigabyte space, are still protected from each other. This means that while Windows 95 is a dramatic improvement over Windows 3.x, it is nowhere near as secure as Windows NT.

Conclusion

The subject of process spaces can be complex and this is a somewhat simplified version of what is going on behind the scenes. It did not, for example, discuss how 16 bit application memory is managed, or what happens when an application tries to write into a dynamic link library that is mapped into several applications (which is possible under Windows NT, but is, believe it or not, safe due to a built in copy on write mode). Chapter 14 in the VB Programmer's Guide to the Win32 API goes into substantially more detail, as it takes the next step and explains how to use the many new API functions that are designed to work with virtual memory spaces.

Extending the Capabilities of the Windows Common Dialogs

How many times have you opened the File Save dialog box only to have to switch to the File Manager in order to create a new directory in which to save the file? Or how many times had you opened the File Open dialog box and could not remember the name of the file that you worked on last Friday. Or had to switch to the File Manager in order to look at the last modification date of the file you want to open? If you have encountered these annoying limitations in other programs chances are that your customers have also encountered them in yours.

Common Dialogs are Good

Common dialogs are a strong step in the right direction. They standardize the "look and feel" of certain operations and provide some useful functionality - all implemented in a stable and thoroughly tested DLL. Though you may not realize it based on the common dialog support provided by Visual Basic, common dialogs were designed to be extended or customized with relative ease. You can frequently add quite a bit of functionality without jeopardizing the reliability of the control or violating the user interface commonality that makes them so valuable. There's one catch - in order to modify the common dialogs, you must refer to API functions inside the common dialog library, COMMDLG.DLL - you can't just use the cmdialog.vbx control that comes with Visual Basic. Cmdialog.vbx is just a wrapper for the commdlg API functions, and it does not expose the hooks that are necessary to modify the dialogs.

You ~~Can't~~ Can do that in Visual Basic - with SpyWorks

SpyWorks comes to the rescue once more with its CBK.VBX control and SpyNotes #2 - The Common Dialog Toolkit. The CBK (callback) control allows you to provide a callback function address when calling the COMMDLG.DLL library functions directly. *The Common Dialog Toolkit* shows you how to modify each of the common dialogs. When a dialog box and all its controls are loaded, a WM_INITDIALOG message is sent to the dialog box (which the callback function receives). At this time, you may make any additional initialization or modifications to the dialog box or to any of its controls before it is displayed. Among the things you can do are: position or re-size the dialog; change the dialog's caption; and hide, move or add controls. Here is an example on how to position the dialog box relative to its parent window at load time. You need to know the window handle of the parent window and the x and y offset position from the parent window in pixels. Create a point structure type that holds and x and y integer and initialize them to the x and y offset positions. Call the ClientToScreen API function to retrieve the screen coordinates of the offset position, this is where the dialog will be positioned. Call the Move Window API function to position the dialog at that location.

```
Dim rc As RECT
Dim pt As POINTAPI

pt.x=xoffset
pt.y=yoffset
ClientToScreen hwndparent, pt
GetWindowsRect dlgwnd, rc 'get dlg box dimensions
'move window to the specified coordinated of the parent window
'you can also resize the window if you specify a value for lengthoffset and
widthoffset
MoveWindow dlgwnd, pt.x, pt.y, rc.right - re.Left + lengthoffset, rc.bottom
- re.Top + widthoffset, 0
```

Other Cool Things for Your Common Dialogs

Add a "Create Directory" command button in your FileSave common dialog. Hide the "Network..." button in your FileSave or FileOpen common dialogs. Display additional file information (such as file size, last modification, date, etc.) each time the user highlights a file name from the File List box. Add an Image control to the File Open dialog so that you can preview picture files before opening them. Make the Font or Color dialogs modeless so that you can apply the font or color you choose and see the results without closing the dialog box. Add a font text preview text box into the Font dialog so that you can preview a specified text string (instead of "AaBbYyZz") when selecting font attributes. Initialize the Printer Setup dialog's Orientation to Landscape, or PaperSize to Legal without changing the system settings for the printer. Make your common dialogs three dimensional.

Learn More about SpyWorks and Windows with the Common Dialog Toolkit

The Common Dialog Toolkit provides plenty of documented sample code on using SpyWorks controls, Windows API functions and other techniques. One of the goals for the SpyNotes series is to educate SpyWorks users on processing some of the basic Windows messages. Another is to show examples on how to use some of the API functions included in dwspydll.dll, and provide more advanced technical information on Windows and SpyWorks. The *Common Dialog Toolkit* can go a long way towards helping you use these and other advanced Windows programming techniques.

Classes With StorageTools

One of the great new features of Visual Basic 4.0 is the ability to create classes. Classes let you encapsulate private data and public methods for operating on that data. This allows you to package functionality into a reusable unit which can be used without knowledge of how it is actually implemented. Unfortunately, the barrier between a class and the program that uses it is broken when a class has to read or write any information on a disk. Either the class writes its information into a separate file - resulting in a mess - or the class has to be specifically written to write into the program's file format - resulting in an non-reusable class that has to be changed whenever the file format is changed.

A neat solution to this problem can be found in every copy of Desaware's StorageTools. StorageTools includes two Visual Basic 4.0 projects demonstrating examples of class usage with the Storage OLE control. Class.VBP is a OLE DLL class that controls a single graphic figure. ClsTst.VBP is a simple program that uses a number of Class objects. ClsTst writes each picture to a single Structured Storage file using the Storage control. However, each individual figure is saved by that instance of the class. This is done by passing a single stream to each instance of the class. The class then saves its information to that stream.

There are a number of advantages to this method. First, streams within a Structured Storage file can be any size without affecting any other stream. You could easily modify the figure class to write more or less information without changing any other part of the project. Second, streams are independent parts of the Structured Storage file. The class will write to any stream that is passed to it, whether it is in the root storage of a tiny file or 4 levels deep in a much larger one. Changing the entire architechture of the file would not affect the class at all.

Visual Basic User Groups

If you are a leader of a Visual Basic User Group and would like some promotional materials for some of your meetings, please contact Roan Bear in our office. Or just drop us a note and let us know who and where you are!

You can request copies of the current newsletter or special promotional software. Please give us 2-3 weeks notice in requesting materials for your gathering. Thanks!

Persistence of Data

Every application works with data, and most need to save that data between sessions. The term "persisting data" is often used to describe this. Before looking at some of the techniques that are available for persisting data, it is worth considering some of the reasons for doing so.

- You may need to save configuration information for an application.
- You may need to save configuration information for an application on a per user basis.
- You may wish your application to have one or more associated documents. For example: A word processor has document files. A spreadsheet program has spreadsheet files.
- You may wish your application to work with information in a database which can hold multiple records of information on objects that are similar to each other.

There is no one data storage technique that is ideal for all of these purposes. You probably would not want to use a database to save application configuration or setup information. You would not use a private initialization file to save thousands of complex records of the type typically found in a database.

It is not possible to come up with an absolute set of rules for which techniques are appropriate for which applications - certainly not in a short article here, but it is possible to come up with some general guidelines.

For general configuration data, including current settings for an application, a private initialization file or the system registry are appropriate. Initialization files are somewhat easier to use and allow for easy user editing. The registry supports more data types and due to its hierarchical nature is better for per user configuration data.

For docucentric applications, avoid the temptation to use a database. Databases are designed to support multiple records where each record is similar. They are not generally appropriate for documents that contain arbitrary graphics, text and binary data. This is due to the high overhead and programming complexity required to implement this type of document with a database. Instead, docucentric applications should usually use private file formats that are read and written directly. OLE structured storage can be used to simplify the task of organizing complex data within a document. Desaware's StorageTools makes it possible to use OLE structured storage from within Visual Basic.

Applications that need to store sets of data that are similar to each other will typically use databases. This is especially true if the data needs to be accessed by more than one application at once or if searching, filtering and report generation is required.

Introducing - Desaware's new API Class Library

Visual Basic programmer's have long been aware that you can substantially increase the power of your Visual Basic applications by taking advantage of the hundreds of functions that are built into the Windows API. Desaware has a long history of helping programmers take advantage of API functions, first through the Visual Basic Programmer's Guide to the Windows API, and now through the forthcoming Visual Basic Programmer's Guide to the Win32 API.

API functions are powerful, but they can be difficult to use. One way to help programmers learn and use API functions is to encapsulate them into a function, class or object library. Desaware's new API Class Library takes a rather unique approach in this regard. You see, there is no one best way to access API functions. Sometimes it is best to use a standalone API object library. Sometimes it is best to incorporate a class library into your application. Sometimes it is best to add inline code into your own program. By including complete source code, Desaware's Class Library supports all of these approaches. Cut and paste code into your program. Add the classes into your application. Or compile the classes into your own API DLL server - the choice is yours.

The Desaware API Class Library will be part of the February release of SpyWorks Professional, and a partial version is included with the Visual Basic Programmer's Guide to the Win32 API.

SpyWorks 4.0 for 16 and 32 bit OLE Containers

The Standard Edition

The original SpyWorks-VB package became one of the most successful Visual Basic add-on products of all time by allowing programmers to do virtually anything in Visual Basic that is possible with other languages. It does this by supporting powerful Windows programming techniques that are not part of Visual Basic.

SpyWorks 4.0 extends this philosophy to the new world of 32 bit programming for not only Visual Basic 4.0, but other full OLE control containers as well. It does this by providing easy access to low level system capabilities such as subclassing, callbacks and hooks that are rarely built in to higher level programming systems such as Visual Basic and VBA applications. In fact, SpyWorks makes these tasks so easy, that it will often be more economical to use SpyWorks for these tasks even with languages such as C and C++!

Version 4.0 standard edition is a 16 and 32 bit OLE Control based edition of the SpyWorks series.

The Professional Edition: A subscription that Includes 3 free updates!

We know that OLE controls are a new technology, that the containers that you will be using such as Visual Basic 4.0 have only just appeared, as have the operating systems that you will be using: Microsoft Windows® 95 and Windows NTTM 3.51. What new capabilities are built into the operating systems that SpyWorks can help you utilize? What kind of features will your customers demand? There is a lot to learn, and the Professional Edition is intended to not only help you learn it, but to evolve to meet your needs. Every four months you will receive a complete update of the entire SpyWorks series, 16 and 32 bit OLE Control and 16 bit VBX along with the latest utilities, application notes and so on. We will be adding features to the controls based on the most often requested (and most intriguing) features. **Plus, it includes the full SpyWorks 2.1 package and Common Dialog Toolkit.**

SpyWorks-VB 4.0 Custom Controls:

The DWSBCxx.ocx subclassing control provides you with complete access to the underlying Windows message stream for the forms, controls and other windows in your application or other applications.

The DWSHKxx.ocx hook control allows you to take advantage of Windows hooks. Detect messages for groups of controls, for entire applications or for the entire system. Detect keystrokes on a task or systemwide basis to create “hot keys”. Use journaling to create macro recorders or computer assisted training sessions - or to monitor user interaction to test your user interface. The applications are limitless.

The DWCBKxx.ocx control allows you to use windows callbacks. These are Windows API and DLL functions that require a procedure address in order to work.

The DWEASYxx.ocx control provides an easy to use implementation of common subclassing tasks including virtual forms, MouseEnter/MouseExit detection, rollup windows, caption buttons and much more.

The DWSPYxx.dll dynamic link library provides an array of library functions designed to help you take full advantage of both the Windows API and the SpyWorks controls. It also includes indirect property access capabilities, which let you programatically access properties of other forms and controls in your application.

Other SpyWorks-VB 4.0 Features:

The first release of the professional edition includes 32 bit OLE controls and the SpyWorks 2.1 VBX based package. 16 bit OLE controls and additional utilities and application notes will follow in the second release currently scheduled for January ‘96. The standard edition, containing 16 and 32 bit OLE controls only, will ship in that time-frame as well. **As always, applications created with any SpyWorks edition may be distributed without royalties.**

SpyWorks-VB 2.1 for Visual Basic 3.0 & 4.0(16 bit)

SpyWorks-VB version 2.1 is latest VBX based edition of SpyWorks for 16 bit Visual Basic. With new features and full support for Windows 95, it is ideal for those VB programmers who are not planning an immediate transition to 32 bit Visual Basic (for those that are, SpyWorks 2.1 is included in the professional edition of SpyWorks 4.0).

You CAN do it in Visual Basic

We can barely begin to describe all of the possibilities provided by SpyWorks-VB, but here are a few: Create VB add-ins to other applications or VB itself using cross-task subclassing · Printer configuration · MouseEnter/MouseExit for status bars · Tiny Captions and Rollup Windows · Scrolling Forms/Controls · True LostFocus/GotFocus detection · Create owner draw controls · Trap tab and alt-tab keys · Use Windows enumeration functions · Detect WIN.INI changes · Private and owner draw clipboard formats · Change the behavior of standard controls · Indirect property access · Obtain an event history for a program · Detect Windows API parameter errors · Many data conversion and low level access functions · Free SpyNotes #1 advanced application notes · and more....

SpyWorks-VB 2.1 Custom Controls:

SpyWorks-VB 2.1 contains VBX equivalents to the OLE controls defined above, plus a Visual Basic aware set of system utitlities and debugging tools.

SpyWorks-VB 4.0 Professional is $249. 4.0 Standard and 2.1 are only $129. For Microsoft Windows 3.1x, Windows NT and Windows 95

StorageTools Version 1.0

The ultimate data storage and file manipulation toolkit! Change the way you work with files and data forever!

You may have heard about OLE 2.0 with its ability to control applications and embed objects from one application into another. But there's another feature built into OLE 2.0 that until now was not accessible to Visual Basic programmers. It's called "Structured Storage" and StorageTools is your key to this powerful technology.

StorageTools allows you to take advantage of the same file storage system used by Microsoft's own applications. Includes sample programs (with source) that let you examine the structure of any OLE 2.0 based file so you can see exactly how they do it!

StorageTools Storage Control:

The Storage control uses Visual Basic OLE Objects to represent storages and streams, creating a familiar interface for manipulating those elements. Within the Structured Storage standard are a number of functions that let you upgrade your file handling procedures easily: you can detect whether files are in Structured Storage format or not, and you can read any normal file as if it was a Structured Storage file. The Storage control also adds a number of features specifically for Visual Basic programmers. It lets you read and write information in any of the styles Visual Basic uses, such as "Sequential", "Binary" or "Random", including arrays.

StorageTools is the key to unlocking the following Structured Storage capabilities:

- Utilize a powerful new method for organizing information within a file or compound document.
- Buffered file modifications allow you to easily undo modifications to a file using Transactioning.
- Expose important information to outside applications by including a standard document information stream.
- You can organize information in memory exactly like you organize information on disk. Only one function call is required to copy information between a Storage in memory and a Storage on disk.
- Reduce disk access time by only saving the parts of the document that have changed.
- All disk space allocation and organization are done for you.
- **And more....**

StorageTools Registry Control:

The Registry Control makes it easy to access the Registry and the Registration Database. It provides all the power of Windows API calls and more. Keys within the Registry are accessed like directories. Values are converted to data types compatible with Visual Basic. Powerful search capabilities are included, allowing you not to only search among keys and value names, but within the value data itself. The documentation includes descriptions of areas in the Registry that might be useful and how to access them.

Using StorageTools - the Registry becomes the key to transforming your Visual Basic program into a truly professional application. Here are just some of the thing you can do:

Register the filename extension to your documents, so your program is automatically launched when a user activates one.

- Create user-specific and machine-specific configuration settings, dramatically improving your user interface.
- Expose DDE links to your program.
- Interface with Windows 95: set icons, register an uninstaller, update reference counts for DLL's you use, and much more.
- Expose your program's version information.
- Read and modify system settings.
- Includes 16 & 32 bit OCX for Visual Basic 4.0 and other OCX containers. Only $129. Supports Windows 3.1x, Windows NT and Windows 95

VersionStamper version 4.0

Reduce Technical Support Costs!

An end-user or customer calls. Your program is behaving strangely, or no longer loads. Did another program install an older DLL, VBX or OCX on their system? Or maybe a later, but incompatible version? Is it a conflict due to an incompatible DLL already loaded in memory? Has the PATH environment setting changed? Is the file properly registered in the registry? Is a required file missing? Now your program could tell you what's gone wrong... with VersionStamper.

Detect OCX, VBX and DLL Incompatibilities Instantly!

VersionStamper allows you to easily embed into your Visual Basic executable or other OLE container information about all of the dynamic link libraries and custom controls used by that application, along with required current versions, file date and time, and your choice of warning conditions. The VersionStamper OLE custom control can use this information to check the versions of the actual modules that are available on the target system. ***Incompatibilities are instantly detected!***

VersionStamper is designed with your needs in mind. You can perform version verification automatically on load, or at any time you wish under program control, or even dynamically verify a list of files at run time. Plus, a customizable and redistributable "rescue" program can be used to detect and log problems when an application won't load at all. VersionStamper is designed primarily for DLLs, OCXs, and VBXs, but will work with any type of file, even those without embedded version information.

A COMPLETE Solution and Tool Kit.

VersionStamper is easy to use even if you are an absolute beginner - but we didn't forget the experts. Look what else you get:

dwvstp??.ocx - The 16 and 32 bit OLE control includes extensive dialog box support for embedding version information for selected DLLs, OCXs, VBXs and other files. Each control can embed two file lists so you can use a single control to create dual platform executables. It also has the ability to scan Visual Basic 4.0 -based projects to automatically create a list of DLLs, OCXs, and VBXs required by the application. The OLE version even allows you to scan for files not specified in your file list during run-time. The run-time distributable control is royalty free. **These controls are fully compatible with Visual Basic 4.0 and other true OLE Control containers!**

- **dwvstamp.vbx** - This version stamping custom control has extensive dialog box support for embedding version information for selected DLLs, VBXs and other files. This VBX version can also embed true version resources into Visual Basic version 3.0 executables.
- **VerInfo.exe** - A Visual Basic-based program that allows you to quickly view the version information for any executable, DLL, OCX or VBX that contains versioning information. Includes source code. It also produces a complete list of the files required by the application based on information embedded by the **VersionStamper** control.
- **VerResQ.exe -** A Visual Basic-based program that analyzes any Visual Basic-based executable stamped by **VersionStamper**. It instantly detects any incompatibilities that exist between the requested files and those actually present on disk. Includes source code so that it can be customized for distribution with your application.
- **VerSplsh.mak** - This sample project demonstrates using a standalone executable to perform file verification for your main application. It displays a splash screen during file verification. If no conflicts were found, it Shells your application, otherwise a form is displayed notifying the user of the conflicts. This method will allow you to identify any incompatibilities that exist - even if the executable cannot be loaded. Designed to be easily modified so that you can create your own splash screen shell program to distribute with your application.
- **VsRTDemo.mak** - This sample project demonstrates a new feature of **VersionStamper** (OLE control only) - how to scan files dynamically during run-time.
- **Technical Notes** include a complete discussion of the Windows-based system of version stamping and the embedding techniques used by **VersionStamper**. It also includes detailed suggestions and examples for improving the Visual Basic setup toolkit. Version 4.0 extends includes an extensive

discussion of platform dependent issues including Windows 95 and Windows NT.

Whether you are developing for corporate use in house, or for distribution worldwide, **VersionStamper** will pay for itself the very first time a customer calls with a compatibility problem. You'll have a solution in seconds - with **VersionStamper**.

Includes 16 & 32 bit OCX for Visual Basic 4.0 and other OCX containers. Includes VB 3.0 VBX version which can embed true version resources into your VB executables. Only $129. Supports Microsoft Windows 3.1x, Windows NT and Windows 95

Newsletter Bonus Disks

The first three newsletters had bonus disks associated with them which contained all of the sample programs. These samples can be found in the **\Articles\VBBultn\Source** directory on the CD-ROM for the Visual Basic Programmer's Guide to the Win32 API.

The samples in this directory are Visual Basic 3.0 programs. Some of them are not compatible with 32 bit operating systems such as Windows NT and Windows 95. Updated samples for many of these applications are included with SpyWorks 4.0. One or two 32 bit examples can be found in the **\Articles\VBBultn\Source32** directory on the book's CD-ROM.

Most of these samples require that you be a licensed owner of SpyWorks 2.1 or VersionStamper 1.0 or later in order to run in design mode. However, the executable files will run using the runtime controls and DLL's included in that directory.

The Desaware Visual Basic Bulletin

Volume 2 Issue 2

Introduction

Welcome to the third issue of our newsletter. I think this is our most exciting issue yet, largely due to the enormous changes going on now in our industry. Windows 95, Windows NT, Visual Basic 4.0, Office 95, Delphi - how can one keep up?

I've always felt that one of the ways that Desaware distinguishes itself as a vendor is that we are not just in the business of selling components and tools - we are really in the business of teaching programmers what they need to know to do their jobs - to create the best applications possible. Part of that teaching is to provide tools that are not your typical "canned" components, but rather tools that extend the capabilities of the programming environment itself.

For the past year and a half, I've spent most of my time studying (as have the other developers at Desaware), focusing on two areas - 32 bit Windows, and OLE controls. In this issue of our newsletter, you'll see some of the results, with more to follow over the next few months.

One of the things that we are trying follows my philosophy of attempting to provide information along with product. The SpyWorks professional subscription edition addresses a problem we ran into with the original SpyWorks - there is great demand for more information on how to use the product, yet there is no good, economical way to provide it. By offering a software subscription, we can provide to those customers who need it, not only articles and application notes, but new product features that are influenced in a large part by the demands of our subscribers.

This newsletter issue may have a differerent feel than the past two (if not a different look - I did say we were trying new things!). Instead of a lot of shorter how-to articles, there is a greater emphasis on some of the underlying technology with which we have been dealing over the past year. In the next issue, the focus will return to emphasizing practical tips and techniques. The issue is four pages larger as well, to allow us to include new product information in the newsletter itself.

This is an exciting time for us as I'm sure it is for you as well. For those of you who are already Desaware customers, thank you for your support through the Visual Basic 1.0 through 3.0 generations. I think you'll find us a good source of support in the transition to Visual Basic 4.0, 32 bit and other development platforms. For those of you who are not yet Desaware customers, I invite you to get to know us through our newsletter, online sources or just give us a call. Our approach, style, marketing and types of products may not be typical, but in customer service, product quality and level of technology, I believe (with all due modestly) that we are among the best in the business.

Daniel Appleman

Inside…

The SpyWorks Series
Current Affairs

Product Information:

(Note: These are the original product descriptions as they appeared in the newsletter. For more detailed information, refer to the Desaware online catalog)

SpyWorks 4.0 for 16 and 32 bit OLE Containers
StorageTools Version 1.0
VersionStamper version 4.0
The Visual Basic Programmer's Guide to the Win32 API
SpyNotes #2 The Common Dialog Toolkit
Classics from Desaware - CCF-Cursors and The Custom Control Factory
PC Magazine's Visual Basic Programmer's Guide to the Windows API
How Computer Programming Works

Jump to Newsletter #6
Jump to Newsletter #5
Jump to Newsletter #4
Jump to Newsletter #2
Jump to Newsletter #1

Structured Storage

Practically every program needs to read and write information to disk. In some cases, the solution is to use a database system. But there are many cases where a database is not appropriate. Sometimes the data that needs to be stored does not fit into the structure of a database - for example: if you were creating a word processor you would certainly not use a database to create document files. In other situations, you might not want the overhead of a database engine in terms of memory, performance, the need to redistribute large database components or require that your end user have the same database system. For all of these cases, a private file format is a more logical approach.

This means that practically every programmer has the problem of figuring out how to organize information within a file. This can be a complex task because of the nature of sequential files. Consider the case where you have to add a single character near the start of the file. This task requires a significant amount of time because even after changing a single character, you have to rewrite the entire file - a substantial performance hit, especially as files get larger.

And of course, the file format would, most likely, be completely incompatible with those of other programs. To retreive even partial information from a file requires intimate knowledge of the entire file format. For example, to find out how many pages are in your word proccessing document, you must load the program that created it. As your file size increases, performance will degrade as you find it necessary to rearrange blocks of data within the file.

In order to overcome these problems, Microsoft has devised a new standardized technique for structuring blocks of data within the confines of a single file. This standard is called Structured Storage, and is implemented as part of the Object Linking and Embeding (OLE) subsystem. It allows you to organize blocks of data within a file just as you would organize files on a drive - in a heiarchical structure. The elements analogous to directories are called “Storages” and those analogous to files are called “Streams”. A program familiar with the Structured Storage format can browse the contents of any Structured Storage file without having the slightest clue as to which program created the file or what the file contains. And by including a stream containing a standardized data structure, a file can expose useful information to any program that recognizes the standard. A file that follows these standards is also called a “Compound Document”. Microsoft Word 6.0 follows this example. It saves its documents in Structured Storage files, and includes a common information stream. When the properties of a Word 6.0 document are viewed in Windows 95, it shows such information as title, author and page count. Again, Windows 95 does not need to know the format of a Word document to do this, and this capability is available to any application.

Because the information in a Structured Storage file is stored in separate blocks of data, it is no longer nessessary to write or read an entire file to adjust for minor changes. By saving each page in its own Stream, for example, you need only write the page that was modified. An element within the file can be added, deleted, moved or copied while all other elements remain untouched. This makes it possible to easily perform incremental saves of information in files instead of having to rewrite the entire file each time a change is made. All disk memory allocation is done for you, dramatically simplifying your job as a programmer.

You can even create a Structured Storage within a block of memory. This lets you store information in memory just as you would store it on disk. The Structured Storage system handles all memory allocation for you. In Windows 3.1, you can share this block of memory between programs, creating a powerful method for sharing large amounts of complex data.

Because the Structured Storage uses a different standard than normal DOS file access, any number of storages and streams can be kept open at the cost of only one file handle per Structured Storage file.

Structured Storage implements one other incredibly powerful technology called “transactioning”. If you open a part of the file in transactioned mode, you can make as many changes to it as you wish. At any time you can decide to “commit” those changes and make them permanent, or “revert” them - i.e. cancel them - regardless of how many changes you have made. This makes it possible to easily add “undo” functionality to your applications that work with files.

Desaware’s StorageTools exposes all of the capabilities of OLE strucutured storage to Visual Basic programmers, bridging the gap between VB’s traditional (and limited) sequential and binary file access, and the use of a full database system.

File Verification under Win32

There are several important changes in Win32 that affect the order applications use to load components. First of all, 32-Bit applications are not required to use a file already loaded in memory by another application. 32-Bit applications may load the file from the same or different path because shared files are no longer shared across all applications in the system. Instead, the shared file is loaded into your application's process space, making it actually part of your application. Second, the registry (or registration database) is used more extensively. The registry replaces the INI files, it is central depository containing all sorts of information. Most OLE controls are expected to register itself into the registry. After being registered, the path and other information for the file would be available to any program that queries for that component. Thus, applications that use a certain component would know where to load the file from.

So, what distribution problems do these changes solve, and what new problems do they introduce?

The ability for 32-bit applications to map a component into its own process space eliminates one of the most frustrating problems dealing with shared components. You don't have to worry anymore about other programs already running with older versions of a component trashing your application. Your application will still check first to see whether the component is already loaded in memory, but only within its own process space, not globally. If the component is not loaded, it will go through the normal search to find the required component and map that component into your own application's process space. One thing to be aware of is that it is now possible to load more than one version of the same dynamic link library (DLL) into your application, a dangerous thing to do if you are not careful.

In Windows 3.1x, components are often stored all over the place. They can be found in the Windows directory, Systems directory, private directories, and directories in the user's path. If different versions of the same component exist, the file that will be loaded may depend on the current directory, or the path environment. Under Win32, files can still be stored in many different directories, but Microsoft introduced the registry to provide a central location for programs to query for the path of a particular file. New components should register itself in the registry so applications that use it can query for the directory to load the file from. Unfortunately, you have no control over what's registered in the registry after you register a component. Other applications can unregister the required component and register the wrong component in its place. Once a component is registered, it must exist in the registered directory with no exceptions. When an application queries the registry for the path of the component, it assumes that it exists in the specified path. If the application does not find it in the specified location, it will not attempt to search for that component anywhere else. In Visual Basic version 4.0, you get an "Error Loading File" message box and it doesn't tell you which file caused the error. The registry also proves to be a very intimidating place to browse manually. Using the Registry Editor to browse the registry for an unfamiliar component may take some time and patience.

Additional file types

So far, what we've discussed focus mainly on DLLs, VBXs and OCXs. There are many cases where applications are also distributed with certain types of databases, licensing files, other executables, and other application critical files. These files must all be in "sync" in order for the application to run properly. VersionStamper is not limited to just checking for DLLs, VBXs, and OCXs. You can embed any type of file into VersionStamper's custom control to check for.

Conditional Verification and Multiple File Lists

VersionStamper allows you to run a verification while your application is loading or at any other time. It can even verify your application before it is loaded, allowing you to catch potential conflicts and maybe even correcting them before your application starts. Each VersionStamper control can embed two list of files. This is ideal when writing applications that will run under both 16 and 32-bit Windows. You can create a list of files based on the files required for the 16-bit version, and a list of files based on the files required for the 32-bit version. You can create additional file lists by adding more VersionStamper controls to your project. There are other situations where the files used by an application may depend on different circumstances. For example, some features of an application may be disabled for a particular user if that user does not have the required hardware or license. Rather than embedding all

the different combinations of file lists into your application, you may want to dynamically create a file list during run-time based on information about the user. VersionStamper allows you to specify the files to verify at run-time.

Conclusion

Using components under Win32 has improved a bit, but many conflict problems can still occur. VersionStamper cannot solve file conflicts, but it does notify you to when they occur and can save you a lot of customer support calls.

The Registry & Why You Would Want To Use It

One of the key features of Microsoft Windows is its ability to be customized to meet the needs of each user. Not only can a user specify color schemes or pictures on the desktop, but programs can (if the programmer so specifies), remember their last location and which documents were most recently used. Since the first version of Windows, Microsoft has recommended that applications store this type of configuration information. To facilitate this task, Microsoft created a common place for this information to be stored: a text file called WIN.INI. Despite a tendency for the file to become hoplessly cluttered, it was sufficient for the first versions of Windows simply because there were so few Windows applications. When Windows 3.0 became popular, the limitations of this single configuration file became apparent. Windows 3.1 introduced the idea of private initialization files - INI files that were private to a single application, or related group of applications. This caused new problems. First, the WINDOWS directory became cluttered with .INI files. Second, if for some reason users wanted to change a configuration item, they would have to wade through pages of unorganized values. Finally, it became difficult for programs to learn about or to modify other programs. With the introduction of Dynamic Data Exchange (DDE) and Object Linking & Embedding (OLE) - both of which are essentially technologies for comunication between programs - it became clear that a new standard for configuration storage would have to be designed.

The result was the Registration Database. You can explore it yourself with the REGEDIT.EXE program that should reside in your \WINDOWS directory. It was designed to overcome some of the problems that resulted from the explosion of .INI files. It was a single database that was supported by Windows and was accessable to all programs. Information that should be commonly available (such as descibing DDE links, or describing types of files based on extensions) could now be kept in a central repository. Information could be stored in heirchical form, allowing complex relationships between values to be stored, as opposed to the flat organization of .INI files. The database itself was in binary form, allowing quicker access. However, it was still limited to 64K in size, and it was not meant as a complete replacement for .INI files.

With the creation of Windows NT, a new, massive, powerful, centralized configuration system was developed. It is called the Registry. While based upon the Registration Database, the Registry does far more. It completely replaces the WIN.INI and SYSTEM.INI files, and is the recommended location for storing other program's configuration information. Values placed in the Registry it can be in a variety of formats, unlike the Registration Database or .INI files, which can only store strings. Any single value can hold up to a megabyte of information, and the Registry itself can hold more information than you'd ever care to put into it. It is implemented as a number of separate binary files called "hives". These hives are formatted and specially buffered to allow speedy access.

Windows 95 also uses a Registry very similar to Windows NT's, although the operating system does not depend on it nearly as much. Although the WIN.INI file exists, and is used, it can be deleted without harming Windows 95's ability to operate.

While it might take some work to convert your program from using initialization files to using the Registry, you will gain a number of advantages. The Registry is often faster and more robust than accessing a large text file, and registry entries are less likely to be deleted by accident than a text file. The Registry lets you easily store separate configuration information for each user. It is the key for integrating your program with the Windows 95 shell.

While the Registry can be accessed using API calls, Desaware's new StorageTools contains a new OLE control that uses a file-system metaphor to access the Registry - an approach that is both familiar to Visual Basic programmers, and far easier to use. But regardless of how you choose to work with the new system Registry - your applications can't afford to ignore it. It forms the foundation for both system and application configurations under both Windows NT and Windows 95.

Upgrades Upgrades Upgrades

SpyWorks 2.0 to 2.1
VersionStamper 1.0 to 4.0,
Custom Control Factory 2.0 to 4.0

First the free stuff. If you purchased SpyWorks, VersionStamper or Custom Control Factory after August 1, 1995, you are eligible for a free upgrade to version 2.1 or 4.0 respectively. To take advantage of this offer you must send us a copy of your original sales receipt (via mail or fax) with your order. Please be aware that there is a shipping/handling charge of $7.50 per order (U.S. and Canada orders) or $15.00 (international orders) . This offer is valid until January 31, 1996 and available only from Desaware.

For those customers who registered for our free Custom Control Factory upgrade earlier this year, your free copy of the software will be sent automatically. Please contact us, if you have not received you software by December 1, 1995.

Now for those customers who are not eligible for the free upgrade, we have modestly priced the cost of upgrading to SpyWorks 2.1 at $45, VersionStamper 4.0 at $79 and for Custom Control Factory $24, plus shipping/handling and applicable sales tax.

SpyWorks 2.1 to 4.0 Professional or Standard Edition

Upgrading to SpyWorks 4.0 is a two step process of first upgrading to SpyWorks 2.1 and then to 4.0 Professional or Standard as warranted The upgrade process has been structured in this way to insure that our customers have the most current software.

Our suggestion is to take advantage of the SpyWorks Professional edition which includes the one year subscription service for upgrades. We have priced the upgrade accordingly (and affordably at $165). Take a few moments and review the SpyWorks series article on page 12, we think that you will agree that the subscription service will keep you up to date with a minimum of effort on your part with the latest features and controls.

Is Visual Basic 4.0 Any Good?

(I had the privilige of being one of the two featured speakers for the Visual Basic 4.0 launch at the Microsoft Developer's Days Santa Clara Venue. The following comments formed the basis for my session introduction.)

How does one judge a computer language? You may have already seen articles in the weekly magazines that preview Visual Basic 4.0 and proceed to judge it. Is it better or worse than expectations? How does it match up with the forthcoming 32 bit Delphi? With C++? With other development environments? You will undoubtedly see more of these articles in the months to come.

By what criteria do these magazines make their judgements? Performance is certainly one criteria. Features lists - those ubiquitious lists of features with checkboxes are also frequently used. Still, I wonder if the way the media judges this type of products is not overlooking something.

Allow me to share with you an insight that I gained several years ago when upgrading from Word for Windows 1.0 to 2.0. I liked version 1.0. It was easy to use and straightforward. As an author, I am more interested in content than features, so I rarely used some of the more esoteric features - the "bells and whistles" of the program. When I did need to do something special in the way of operation or formatting, I would look up the information in the manual, figure out how to do it and move on.

The upgrade to 2.0 went smoothly - it was several months before I noticed something interesting. In all of the time since the upgrade I had not needed to use the manual once. Never. Occasionally I would use the online help for a specific task, but even that was highly unusual. At the same time I had started working with Microsoft PowerPoint. Not only did I never crack the manual for that product - I never used the online help either. You see, while everyone was judging these products or upgrades on the features that they provided, they typically failed to see the most important change: these programs were truly intuitive. The real thing - not a marketing buzzword. Someone at Microsoft has figured out how to write software that meets the real needs of users and that is truly intuitive - and hardly anyone has noticed it. This is the real reason that Microsoft Office has become such a success. It's not just marketing - it's great software.

So before we look at the bulletted feature list for Visual Basic 4.0, allow me to apply this insight to what you are about to see. It is my considered opinion that the Visual Basic 4.0 team has created the language that most VB programmers really need. It may not yet be everything they want. It may not match other tools feature for feature. But many of the new features are truly significant, and many the missing ones are irrelavent.

Here's an example: more than one programmer has complained to me that Visual Basic 4.0 does not allow creation of multithreaded applications. In most cases I look at the skill level and experience of the individual and tell them that they should be grateful. Those who hear about multithreaded programming and understand it in principle know that it is a great way to allow an application to perform several tasks at once. Those of us who have done multithreaded programming know that it is often a nightmare of biblical proportions - tough to design properly and nearly impossible to debug. Should Microsoft have included multithreading in a tool for programmers, many of whom are only now starting to understand and write event driven software? You must be kidding. Maybe someday Microsoft will figure out how to make Visual Basic multithread in a way that is easy to understand and debug. Until then - lets be glad they left it out.

But wait, you ask. Who is Microsoft to play big brother and protect us from our own mistakes? Have them put the feature in and let the programmer beware! Ah, you see - you also have fallen in to the trap of looking solely at features. You've forgotten what Visual Basic really is. Here's a reminder: *Visual*

Basic is first and foremost a Windows programming language that is safe, efficient and powerful.

Let's take these one at a time. It is a programming language. Not a database tool, though it does that extremely well. Not a glue for OLE components and other OLE enabled applications, though it does that well also. It is a programming language. You will see that many of the most important new features, from line continuation characters to classes directly address features that a serious programmer wants in a language.

Safe - With all of the changes and improvements, Visual Basic 4.0 has not lost sight of this all important philosophy - Visual Basic does not just let you recover nicely from General Protection Faults and exceptions. It does not let them happen in the first place. Errors in your code do not cause system faults - they cause VB errors, and VB tells you where and why it occured and how to fix it. If you have ever done Windows development using C++, you realize what a miracle this is.

Efficient - Easy to use - how do you measure this? The truth is, I can still write a program in Visual Basic in substantially less time than any other language or environment - period. You can argue features all day - when I need quality software written quickly, Visual Basic is the obvious choice. And since I rarely need quality software written slowly... well, you get my drift.

As for powerful - well, that really is the subject of the rest of this presentation.

Before that - As you look at the new features for Visual Basic and evaluate the product in the context of your own needs, I encourage you to keep this thought in perspective. In Visual Basic 4.0, Microsoft has not lost sight of their vision of what Visual Basic is - a windows programming language that is safe, efficient and powerful. In this context, and for this, as a developer, I am grateful.

Under The Hood: Inside a Subclasser

Subclassing is only a small part of the overall SpyWorks package, but it is perhaps the best known. Part of the reason for this is that subclassing is, itself, a very widely used and well known technology. In fact, there is every possibility that you will run into demo or shareware subclassing controls on book disks or online. This naturally raises the question: “is there anything unique about the SpyWorks subclassers, or is subclassing such a simple and routine task that they are basically all the same?” Before you stake your project on any subclassing control, there are some questions that you should ask and features that you should investigate. The stakes are high - subclassing is by its very nature adds a level of risk to your software, making it much easier for your program to crash itself or even the system, not to mention impair performance significantly. You want your control to be designed to limit and control the amount of risk that you take and have a minimal impact on your application and on the system.

Message Filtering

I once got into a long discussion with someone who wrote a simple subclassing control that triggered a Visual Basic event for each incoming message, in which he suggested that this was not a problem and had no significant impact on performance. It is true that there was no measurable impact as long as there was no code in the subclassing event, but as you can imagine, as soon as code was added the situation changed. The last thing you want is to execute Visual Basic code for every incoming message, most of which you do not need.

A good subclassing control should provide a filter that triggers VB events only for those messages that you specify. The SpyWorks subclassers place this filter at the very lowest level where the hook occurs, minimizing the amount of code that runs for messages that are not in the filter list. In fact, due our unique architecture which uses a separate subclassing engine, messages that are not in the filter list do not even make it to the OLE Control or VBX - they are blocked in the low level DLL.

Multiple Windows

There are cases where you may wish to subclass multiple windows. Every control on a form uses memory and resources both from the system and from the container. It is only reasonable for a subclassing control to be able to subclass multiple windows. Of course, in some cases you may wish to use a hook to intercept messages on a form, task or systemwide basis - a capability that is not supported by subclassers, but which is part of the SpyWorks hook controls.

Posted Detection

When you subclass a message, your code is executed as part of the message processing. This means that your application, and receipt of all further messages by the application, is suspended until you have finished processing the message event. This means that in general you should keep code within message events to a minimum. But what about cases where you don’t really need to process the message - all you need is a notification that the message was received? In this case the ideal situation would be for the subclasser to simply record the message as received and then let it complete processing, while posting the notification so that it is received later by your control. You can then handle the message during the course of regular message processing.

Another important issue is that there are some operations that are not allowed during the handling of certain messages. For example: attempting to change the focus during a focus change message can be fatal to your application. Deferred message processing allows you to completely ignore these concerns, since by the time the message is received you have already exited the critical message processing code.

The SpyWorks subclassers all allow a “posted” detection mode which defers triggering of the message information until normal message processing is taking place. SpyWorks also has a built in self-posting mode which allows the control to post a message to itself for later operation. This allows you to execute a small amount of critical code during the initial interception of the message, then post a notification so that further code can be executed later.

Default Processing Control

If you did not subclass a window, the default window function for the window would execute. Your subclasser should be able to intercept the message either before or after the default window function executes. If you intercept the message before the default function, you should have the option to prevent the default function from executing. If you intercept it afterwards, you should have access to the result returned from the default window function.

Cross Task Capability

What if the window that you want to subclass belongs in another task? Under 16 bit windows, this was not terribly difficult to accomplish (though there are some subtleties to it that can cause designers who take the obvious subclassing implementation approach to fail under certain conditions). But under 32 bit Windows (NT and Windows 95), cross task subclassing is much more complex. The dwsbc32.ocx 32 bit subclasser handles cross task subclassing without any difficulties, and provides you with additional information for each message and support functions that let you easily interpret the contents of out of task messages.

Paranoid Design

Subclassing may be simple in concept, but things can get complicated quickly in a real program, especially in Visual Basic where a window may be subclassed by several different controls or add-ins at one time.

What does this mean? It means that a subclasser must carefully keep track of when it is safe to remove itself from a subclassing chain. It must not only be careful to cooperate with other subclassers (so that they continue to function correctly), but must be paranoid itself - check its own state on each message to make sure that another, less sophisticated, subclasser has not made the function addresses that the initial subclasser utilized invalid. And it needs to do so in a way that takes little or no effort on the part of the Visual Basic programmer.

In addition to protecting you from the side effects of other subclassing that may be in effect, Desaware’s subclassers have a unique architecture that protects you from itself and allows you to subclass as many windows as often as you would like without any concern for these kinds of interactions. SpyWorks is the only package that divides the subclassing operation into two components: a high level OLE control or VBX and a low level subclassing engine (dwspy32.dll or dwspydll.dll). All subclassing is done through the subclassing engine which internally monitors which applications and controls are performing a subclass operation. Under SpyWorks, you can subclass a window hundreds of times, using different configurations and messages, and do so from several different applications at once, and be assured that no matter what you have done - the subclassing engine has actually subclassed the window just once. So from the point of view of the system and other controls, the subclassing engine always appears as a single subclassing instance. It is the responsibility of the SpyWorks subclassing engine to arbitrate between the SpyWorks controls and trigger their events. This architecture minimizes the chance of dangerous interactions that can destabalize your system. This architecture is also used by the SpyWorks controls that perform windows hooks.

Developer’s Motivation

Wait a minute: how can the developer’s motivation have anything to do with the implementation of a subclasser? Besides: the developers are doing this just to make money, right?

It is true that motivation may not have the most direct impact on code implementation, and it is also true that the developers who work at Desaware have mortgage, rent and car payments.

But consider this: The SpyWorks subclassers (and other components) were not created as part of a hobby just to see if it can be done, then posted as Shareware to see if anyone was interested. They were not thrown together in order to demonstrate a few principles in a book and thrown on a book disk to help people feel that it was a better value.

The SpyWorks controls were created first and foremost for use in Desaware's own commercial grade Visual Basic applications. They were created because we refused to accept any limitations on the part of Visual Basic. They were created for ourselves - and we would no more compromise on the quality of our components than you would.

Support

Which brings us to a final, but all important factor for any serious developer. Once you start using a subclasser, things can get complex quickly. It takes a great deal of information to take full advantage of this and other low level techniques. It takes sample code. And sometimes it takes a person that you can call or send Email to to get you over the rough spots. We're very proud of the quality of our technical support, and continue to strive to make it better. We've been know to turn around bug fixes when necessary in a matter of days and sometimes hours (no waiting for the next version - we know that your project depends on our components, and we take that responsibility extremely seriously). Advanced components and developer's tools such as SpyWorks is our full time business. We think that when you look under the hood at a subclasser or any other software component, the company that develops and services that component makes a huge difference.

Dear Marian.....

When will Dan's 32 bit Programmer's Guide be available....

I am sure that as soon as possible will not suffice in this instance. Okay, Dan's book should be released by the end of 1995 or early first quarter of 1996. I know that everyone would like a specific date (next week would be greatly appreciated) but honestly, the man has got to sleep sometime. We have kept Dan fairly busy lately in the office with such things as code and manuals, not to mention trade shows and speaking engagements. In the mean time, we promise to have an excerpt from the book in the next newsletter. Give us a call in December to reserve your copy!

How many newsletters have been published and when is the next one due?

Quite honestly, I need to sleep sometime too! This is newsletter number three. Feel free to contact us if you are missing one of the earlier editions. At this time, (programmers know that nothing is ever set in stone) the newsletter will be published three times a year with the next edition scheduled for (oh my heavens!!) January 1996.

When will new versions of the software be available:

As of this publication SpyWorks 2.1 which is the VBX based upgrade for SpyWorks 2.0 is currently available. Spyworks 2.1 contains the latest VBXs. The OCX (32 bit) version, which is SpyWorks 4.0 Professional Edition, will be available the first week of October in addition to VersionStamper 4.0 and our newest product StorageTools. We will also be offering a software suite which includes SpyWorks 4.0 Professional, VersionStamper, StorageTools, and Custom Control Factory.

What is the difference between SpyWorks Standard and SpyWorks Professional?

SpyWorks 4.0 Standard is the OLE control version of the software, whereas SpyWorks 4.0 Professional contains the VBX and the OCX version. In addition, 4.0 Professional edition is a subscription, so not only are you purchasing the current OCX controls, but you are also going to receive three updates during the course of the year (to be sent approximately every three to four months). We understand that as software developers it is imperative that you have the most current controls and information for use in developing your applications. Upgrading to SpyWorks 4.0 Professional edition will allow you to do just that for just $165.00.

The SpyWorks Series

Those of you who have worked with SpyWorks since it first appeared may remember how the product evolved to meet the needs of customers and address issues that arose as we (and our customers) tried to perform new tasks with the product.

One of the first things that we realized when undertaking this new edition is that in a very real sense we are back at square one. The operating systems we are working with are new. The containers (Visual Basic and others) are new. The tasks that need to be accomplished are relatively unknown.

In an ideal world, we would be able to take a year or so to develop the product after the platforms were stable (that's how long it took to do the original product), but obviously that is unacceptable - Too many people depend upon and require SpyWorks today for their own products.

This tradeoff - the demand to meet customer needs along with the lack of information of what this product ultimately needs to be, lead to a somewhat different approach with this version.

Initially, the 32 bit OCX product will be part of a professional/subscription package - those who sign up will receive updates every 3 or 4 months that add new features and address those issues and bugs that arise under both Visual Basic and other OLE control containers. It will also provide a way to offer application notes and other advanced information as we learn them. Subscribers will be assured of always having the latest of everything - VBX, 16 bit OCX, 32 bit OCX, application notes, newsletters and so forth. At $249, we think this will provide an excellent value for the professional Visual Basic programmer.

Once we have a core set of controls that we are happy with, we will start shipping a standard OLE control edition that corresponds roughly to the current SpyWorks package with all of the core controls but without some of the utilities. This $129 package is targetted for more casual users who need to perform specific tasks but are not interested in taking advantage of new features, new platforms or learning more advanced techniques as information becomes available.

We haven't forgetton those people still using Visual Basic 3.0 or who are focusing on 16 bit Visual Basic programming. SpyWorks-VB 2.1 is an upgrade of the classic VBX based product. The primary focus of this edition is full compatibility with Visual Basic 4.0 and Windows 95. It provides a number of new features as well. This package will continue to sell for $129, and is available to current SpyWorks users for only $45 (or, upgrade to the full professional edition for $165 and receive not only the upgraded SpyWorks 2.1, but the full OLE control product and annual subscription!).
Three different editions of a product is a bit more complex than we would like, but it seemed the best way to meet the needs of our many customers who range from individual college students to Fortune 500 corporate sites.

Finally, for thow who are really looking for a deal, check out our new Professional Developer's Suite which includes the professional edition of SpyWorks, StorageWorks-VB, VersionStamper and Custom Control Factory. All this available directly from Desaware for only $429, a savings of over $100 as compared to purchasing the products separately.

Current Affairs

We've moved...

Just in case you haven't noticed we've moved. Our new address is:

Desaware Inc.
1100 East Hamilton Avenue, Suite 4
Campbell, CA 95008

We will continue to receive mail at our old address through the end of the year, but we would appreciate it if you would take a moment to change your records now.

Our phone and fax number remain unchanged.

Come cruise with us!

Check out our new home page on the web. It includes the latest product information and demos. Our address is:

HTTP://www.desaware.com

You are also welcome to visit our forum section on Compuserve, “Go Desaware”

Welcome to Karyn Duncan

We would like to welcome our newest staff person, Karyn Duncan. Karyn will be answering phones and processing orders (among several other things of course!). Feel free to say hello and welcome her to the company next time you call!

A tearful good-bye to Matt Solnit, one of our summer interns, who is off to U.C. San Diego to study something that can't possibly be as important as working for us.

Special Opportunity:

Now you can own one of the original cards from the card house in the VersionStamper ad featured in the Visual Basic Programmer's Journal & the Visual Basic Technical Journal. These collector items are available for you at a modest price of 5$ per card. Yes, this card comes with a Desaware label on it and proceeds will be donated to the Crippled Children's Society of Santa Clara County. Own a piece of history, and help us get rid of all these cards.....{Please!)

Ordering Info

If you are a registered owner of any Desaware products, you will continue to receive these newsletters when published. All others will receive future issues only if we hear from you - just mail or fax this form back requesting a free subscription.

The Desaware Visual Basic Bulletin

Volume 2 Issue 1

We came up with the idea of this newsletter as a mechanism to provide some advanced ideas and techniques to our customers, along with some product information. One of the reasons that the subscription is free is because we didn't want to be bound to a particular schedule - and not being bound to schedules seems to be going around nowadays. You see, we honestly expected this issue to come out in conjunction with either a new version of Visual Basic or some other container that supports OLE controls. So now we've given up waiting, and here we go full speed ahead into another issue of the bulletin, full of some of the coolest things we've learned how to do since the last one, plus news and a special offer or two.

—Dan Appleman

Inside...

Product Information:

(Note: These are the original product descriptions as they appeared in the newsletter. For more detailed information, refer to the Desaware online catalog)

SpyWorks 4.0 for 16 and 32 bit OLE Containers
StorageTools Version 1.0
VersionStamper version 4.0
The Visual Basic Programmer's Guide to the Win32 API
SpyNotes #2 The Common Dialog Toolkit
Classics from Desaware - CCF-Cursors and The Custom Control Factory
PC Magazine's Visual Basic Programmer's Guide to the Windows API
How Computer Programming Works

Jump to Newsletter #6
Jump to Newsletter #5
Jump to Newsletter #4
Jump to Newsletter #3
Jump to Newsletter #1

Stop VBX/DLL Conflicts using Easy "Splash Screen" Implementation.

In our last issue of the Desaware Visual Basic Bulletin, we discussed in detail the troubles that can occur when an application tries to load an obsolete or incorrect version of a custom control or DLL - a problem that is happening with increased frequency as more applications take advantage of software components. We also showed how VersionStamper-VB gives you the ability to scan your target computers for all the required files and correct versions of those files.

VersionStamper-VB provides a number of approaches that you can take when it comes to performing this scan. One approach is to perform the scan as part of your main application's initialization routines. The problem with this approach is that it can be time consuming - after all, each verification does require a certain amount of disk access and a verification of a file that uses many components can take a while. Another approach is to perform the scan only on user request - by invoking a menu option in the main application or by executing a standalone "emergency rescue" program that is able to scan your executable. This approach is fine for detecting problems, but does little in the way of preventing them.

You can combine the best of these two approaches - using a standalone program to perform a scan before your main application is run. The standalone program approach has one very important advantage over performing the verification from within the main application. When you run a Visual Basic executable, it loads all of the required custom controls before executing a single line of Visual Basic code. This means that before the file verification even begins, you may already be running code from an incompatible file. There may be cases where an incompatible file (or files) will crash your application before the file verification can be performed. The standalone program only needs to load the dwvstamp.vbx control in order to perform the verification for your main application. It can read the file list from your main application and verify the compatibility of those files without actually loading them.

Solving the delay problem

Running a standalone verification program before loading your application is a powerful preventive measure and more reliable than building the verification into your main application, but this still does not address the issue of the delay caused by the verification process. Fortunately, we can turn this delay into a strength by using the standalone verification program as a "splash screen" for your main application. You can create a form that displays your product name, serial number, copyright notices, registered user's name, etc. This form remains visible while the verification is in progress. If the verification passes, this form will shell your main application and unload itself. If it fails, you can display your normal warning and error messages.

Start by designing a form to display during the verification. Add a VersionStamper and Timer control to that form. and set the Timer Interval to 100. Then add the appropriate verification template files to the project (refer to pages 22-24 of your VersionStamper-VB manual). The code to start the verification should be placed in the Timer Event (this allows your form to be displayed before file verification process begins). On success, Shell your main application and unload the form.

Other considerations

Some customers may eventually figure out that the verification program is just shelling your main application. They may think that they can bypass the "splash screen" and run your main application directly instead (they will probably run it and find no warnings or errors). One trick you can use is to Shell your main application with some parameters (undocumented of course). For example, you can Shell your main application with the "/verified" string. During the Sub Main or Form_Load of your main application, you can call the Command$ function to retrieve the command line argument. If it does not match, you can display a warning message to the user telling them that file verification was not performed. Another thing to remember when using this method is to install your "file verification" program and your main application in the same directory. Since VersionStamper searches for required files in a specific pattern (the same pattern used to load DLLs and custom controls), these two

programs must be in the same directory in order for the search to be identical for both.

Customer Comment on VersionStamper

As nice as awards are, hearing compliments from our customers is even nicer. And it's a real treat when the compliment comes from a customer who is another Visual Basic add-on vendor! (it turns out that our products are peculiarly suited for extending the Visual Basic environment itself, so it's not surprising that other add-on vendors use them). I received the following Email message from Brad McLane, one of the lead developers at Pinnacle Publishing, who kindly granted me permission to print his comments:

"BTW, you should know that Version Stamper has paid for itself many times over in reduced tech support cost and customer good will. Most tech support problems with ToolThings were directly attributed to DLL or VBX version incompatibilities on the installed system. Version Stamper sniffed out the problems in no time. Happy (and very impressed) users abound!"

ToolThings (for those who don't know) is a set of 10 productivity tools that can help you with your VB programming. You can reach Pinnacle at (800) 231-1293. By the way, they also publish the Visual Basic Developer newsletter for which I am currently writing a regular column.

—Dan

Hidden Treasures of SpyWorks

When people think about SpyWorks, they most often turn to its powerful subclassing and callback capabilities which are used to extend the power of Visual Basic. But there are a number of less well known tools inside the package that can prove surprisingly useful.

INI File Function Library

Did you know that every message name and value used by sbc.vbx and sbchook.vbx, and every group of messages used by these controls and spymem.exe is entirely user configurable? They are all defined in the file spyworks.ini which contains an initial set of default values. This makes it trivial to add custom messages to SpyWorks which are handled in exactly the same way as standard messages. It also makes it possible to define your own group of messages to be detected when spying on messages, making spymsg.exe one of the most flexible tools of its type available.

In order to make it easy to define your own messages and message groups, SpyWorks includes the program swiniedt.exe, which allows you to edit spyworks.ini. As a bonus, this program includes full source code. The file swiapi.bas provides an intermediate level function interface to the ini file, and demonstrates how to manipulate the contents of any private initialization file. It also shows how to use the Alias command to create type safe declarations to access INI files. Though designed for this particular application, the code is easily transferable to other applications or could be used as the basis for a generic INI file library.

The file swimisc.bas contains additional useful functions, including ‘‘ParseAny-String’’ which can be used to extract substrings from a list of strings (The initialization file functions can return a list of values in a special string format in which each value is separated by a NULL character, and the entire list is terminated by two consecutive null characters).

SpyParam - Find hidden API bugs

One of biggest improvements from Windows 3.0 to 3.1 was the reduction in the number of "Unrecoverable Application Errors". One of the ways this was accomplished was by the addition of parameter validation to most Windows API calls. Invalid parameters that would once cause a system crash are now simply ignored.

While this does help make programs more stable, it can lead to hidden bugs that are difficult to find - since simply ignoring invalid parameters does not solve the source of the problem. SpyParam taps into the internal parameter validation of Windows and reports on parameter errors that occur. Like other SpyWorks components, this tool is designed specifically for VB programmers. It can even trigger a Visual Basic runtime error on the exact line where you call an API function with invalid parameters. SpyParam also allows you to filter out parameter errors that are caused by components such as VBX's and DLL's so that you can focus in on your own code.

Here's an example of SpyParam in use. The SelectObject API function is used to select a different drawing object such as a brush for a specific device context. After you are finished using a brush, you should restore the previous brush. Most people do not check the return value from SelectObject when restoring the previous object. SpyParam can detect cases where SelectObject fails due to an invalid device context handle or an invalid GDI object

```
Declare Function GetStockObject% Lib "GDI"      (ByVal nIndex%)
Declare Function SelectObject% Lib "GDI" (ByVal       hDC%, ByVal hObject%)
res% = SelectObject(hDC, GetStockObject         (WHITE_PEN)) 'res now
contains the previous pen
           :
lres& = MoveTo(hDC, X%, Y%)  'Do some drawing
           :
res% = LineTo(hDC, EndX%, EndY%) 'previous      pen is now lost (this is a
bug due to reuse of a   temporary variable)
           :
res% = SelectObject(hDC, res%) 'selects garbage       into the device
context
```

In this case, the White Pen was selected into the device context and something was drawn. But, the variable used to hold the previous pen was reassigned. This error could cause drawing problems later that would be very hard to trace to this code. SpyParam would detect this problem instantly.

SpyParam Helps While Subclassing

Windows programmers often do not bother to check the return value from the PostMessage function, which indicates (among other things) whether the window handle used is valid. Windows can be created and destroyed by various actions in the system, and are even sometimes destroyed and recreated immediately. Take the case where you are sending a series of commands to a window, then posting a final command. If one of the previous commands destroys the window, then every subsequent command using that window handle will fail.

SpyParam can be used to trap "invalid" parameters passed to Windows API functions such as an invalid window handle passed to PostMessage.

```
Declare Function PostMessage% Lib "user" (ByVal hWnd%, ByVal wMsg%, ByVal
wParam%, lParam  As Any)
```

```
Declare Function PostMessageBynum& Lib “User” Alias “PostMessage” (ByVal hWnd
%, ByVal wMsg%, ByVal wParam%, ByVal lParam&)
hWnd = windowarray(index)
‘if the window gets destroyed or is invalid, this will generate an error
that SpyParam can trap
lres& =  PostMessageBynum(hWnd, WM_COMMAND, wp, lp)
```

Create Status Bar Help

Two application features that have recently become popular are status bars and tooltips. Status bars consist of an area towards the bottom of your application screen that displays a brief description of the control under the current mouse location. Tooltips are the increasingly obiquous yellow messages that popup over a control when the mouse lingers in one spot for more than a few seconds.

While SpyWorks-VB does not directly implement either of these features, it does provide controls that allow you to add these features to your application with very little effort, and with less overhead than a standalone control would use to accomplish the same task.

You'll need two controls to implement status bars and tooltips. For status bars you will need a picture box aligned at the bottom of your main form (or any form for which you want to implement status bar help. This control will typically contain a label control which will hold the status bar text - though you can draw the text directly on the control if you wish.

For tooltips you will need a small form which will also probably contain a label for the tooltip text. This form should be loaded during initialization, unloaded when your application is closing, and shown and hidden as necessary. The form's BorderStyle should be set to 1, background color set to &H0080FFFF (same as label), and the Caption, ControlBox, MaxButton, and MinButton properties set to False. The label's AutoSize property may be set to True.

At run-time, you will need to know when the cursor moves into a control (so you can display the appropriate status bar text), when the cursor moves out of a control (so you can clear the status bar text), and when the cursor has remained inside a control for a specified length of time (so you can display the tooltip). You will also need a way to determine which controls have status text or tooltip text attached. Ideally, the implementation should be modular, and not require you to attach code to events for each control.

It's easy using SBCEasy

One of the most powerful, but often overlooked features of SpyWorks-VB is the SBCEasy control, particularly a feature that we call "MouseTransit". The underlying Windows system does provide support for the MouseMove event that can detect when the mouse is moved over a control (though not all VBX controls expose this to the VB programmer). But Windows does not provide any direct mechanism to determine when the mouse enters or exits a particular control or window. SBCEasy solves this problem - taking it a step further by supporting Visual Basic graphical controls as well. By setting SBCEasy's MouseTransit and MouseTransitNC properties, you can receive notification whenever the mouse enters or exits any control in your application. This approach centralizes mouse detection to a single control per form or application (as you choose). SBCEasy's TransitHctl property identifies the control for which the event has been triggered. SpyWorks-VB's indirect property access capability makes it possible to use this value to read the properties for the control. You could, for example, place tooltips and status bar help text in the control's Tag property and use the dwGetPropertyValue function to retrieve the Tag property for the control. A timer control can be used in conjunction with SBCEasy to implement tooltips.

Updating the status bar

Update the status bar text during SBCEasy's MouseEnter event. Parse the Tag property string to retrieve the status bar help and update the status bar label caption. If this control does not have status bar help text, then the status bar would be cleared.

```
Sub SbcEasy1_MouseEnter(...)
  Dim statusstring$
  Dim iresptr%
  statusstring$ = dwGetPropertyValue(SbcEasy1.TransitHctl, "Tag", iresptr%)
  If (iresptr% = 0) Then
     'return the first substring
     statusstring$ = ParseString(statusstring$, 0)
     statusbar.Caption = statusstring$
  End If
```

Showing and hiding the tooltip

Enable the timer control in SBCEasy's MouseEnter event. The timer's interval should be set to the length of time the mouse is required to remain inside the control before the tooltip is shown. During the timer event, disable the timer, parse the Tag property string of the control to retrieve the tooltip text and show the tooltip form. Use the GetCursorPos api function to get the cursor coordinates so that you can position the tooltip directly below the cursor. Set the form's ZOrder property to make it topmost and use the ShowWindow api function to show the form without giving it focus.

```
Sub Timer1_Timer()
  Dim tagstring$, tipstring$
  Dim tippoint As POINTAPI
  Dim toffset%, loffset%, iresptr%
  Timer1.Enabled = False
  tagstring$ = dwGetPropertyValue(SbcEasy1.TransitHctl, "Tag", iresptr%)
  If (iresptr% = 0) Then
     'return the second substring
     tipstring $ = ParseString(tagstring$, 1)
  End If
  If tipstring$  "" Then
     ' calculate position of the form based on cursor position
     GetCursorPos tippoint
     toffset% = 18: loffset% = -2
     ToolTipForm.Top = (tippoint.y + toffset%) * Screen.TwipsPerPixelY
     ToolTipForm.Left = (tippoint.x + loffset%) * Screen.TwipsPerPixelX
     ToolTipForm.ToolTipLab.Caption = tipstring$
     ToolTipForm.Width = ToolTipForm.ToolTipLab.Width + (4 *
Screen.TwipsPerPixelX)
     ToolTipForm.Height = ToolTipForm.ToolTipLab.Height + (2 *
Screen.TwipsPerPixelY)
     ToolTipForm.ZOrder  'place on top of zorder
     'do not activate the tooltip form after displaying it
     res% = ShowWindow(ToolTipForm.hWnd, SW_SHOWNOACTIVATE)
```

During SBCEasy's MouseExit event, hide the tooltip form and disable the timer.

Variations on a theme

What if you are using the tag property for some other purpose? One of the advantages of centralizing the code into a single event or function is that you can easily maintain a table or list of status or tooltip text. You could, for example, use a database - SBCEasy provides the name of the form and control which you can use for a table lookup. You could place the text in a standalone text file and read it in as needed. You can even hardcode the text if you wish. These programmatic approaches provide one overwhelming advantage over the tag property approach - they make it much easier to internationalize your application, as you can use the Windows API to determine the underlying language and select your status and tooltip text accordingly.

Other considerations

SBCEasy tracks the mouse regardless of whether your application is active or not. This means that if you have another application running which is partially covering your application and the mouse moves into your application, SBCEasy will generate a MouseEnter event. You will probably want to display the tooltip only when your application is active. You can call the GetActiveWindow api function during the MouseEnter event to determine whether you should display status bar help or tooltips.

GetActiveWindow returns the window handle of the active Form or MDIForm unless the SetActiveWindow api function was used to explicitly activate a different window.

SBCEasy's TransitHctl property guarantees that a control handle will be valid only during the MouseEnter and MouseExit events. Control handles do not change, but controls may be deleted at any time. As long as you know that a control will not be deleted after a MouseEnter or MouseExit event, it is safe to use SBCEasy's TransitHctl control outside of these events.

I believe in standards. The fact that Windows applications now share a common look and feel is, I think, a good thing overall - it really does make things easier for the end user. But standards are not a religion and should not be followed blindly. This point is most often proven by Microsoft - that strongly promotes the user interface standard, yet does not hesitate to change it any time they feel like it (for example, when releasing a new version of Office, Windows, or anything else).

Who said that status bars can have only text? Who said that tooltips must be yellow? Perhaps you would occasionally want a red tooltip to indicate a potentially dangerous operation? One of the nice things about the SpyWorks approach to implementing status bars and tooltips is that while they are only slightly more complex than a standalone control would be, they are typically more efficient and infinitely more flexible. Why, you could even bring up a full dialog for some controls instead of a tooltip if you felt it necessary.

Awards

Awards are nice. Let's face it - it's nice to receive acknowledgment from the press that your products are cool or useful (almost as nice as hearing that from customers!). As a reader, you may wonder - who are these people who give out the awards? After all, any magazine or newsletter can invent an award - who is to say whether it means anything?

That is why where an award comes from is as important to me as the award itself, and that is why I was so pleased that VersionStamper-VB won the Windows Tech Journal "Star-Tech" award for 1994. What does this mean? It means that the editors of Windows Tech Journal felt that VersionStamper-VB was one of the six best Windows development tools of the year - period. The best from among ALL windows development tools - not just those for Visual Basic. J.D. Hildebrand, the editor of Windows Tech Journal, is known to be somewhat outspoken, and always honest - so this award means a lot to me. If you'd like to judge his editorial skills yourself, you might want to check out their new VB-Tech journal - a rather snazzy new publication that includes a focus on third party products and component based software. You can call them at (800) 234-0386 (BTW: this mention was unsolicited).

-Dan

Avoiding Problems With Intercepted Messages in SpyWorks-VB

You may notice that occasional General Protection Faults (or other run-time errors) happen when you try to perform complex actions during Subclass or Hook events. The reason this happens is that the SBC.VBX, SBCHOOK.VBX and SBCKBD.VBX all stop the message handling portion of Windows while an event takes place. If the code within the event takes too long, the messages start backing up, and eventually cause an error. Because the amount of time before an error takes place depends on the number of messages being sent (and therefore on the number and nature of other programs being run at the same time), these kind of errors can be frustratingly rare

Usually, this problem can be solved by receiving "posted" messages, which do not require that the message handler stop during the VBX event. This can be accomplished by setting the Type property to "Posted" in SBC.VBX, or by setting the Notify property to "1 - Posted" in SBCHOOK.VBX and SBCKBD.VBX. There are problems with this method. First, message parameters can no longer be changed. Second, messages can no longer be deleted before being used. Finally, the message is delayed, and may come too late.

A way around this dilemma is to use the PostEvent property and DelayedEvent event in the SBC.VBX control. As an example, we can imagine a user trying to perform a time-consuming editing function whenever the user presses the "backslash" key. The function will not work in the time constraints required by the KbdHook event, but if a posted method is used, the backslash key cannot be prevented from being sent to the text editor. The solution is to split the code in two parts: the quick code that needs to execute immediately (checking that the key received is the correct one and discarding the key) is put in the KbdHook event. Then an identifying long integer is placed in the SBC control's PostEvent property.

```
Sub HookKbd1_KbdHook (keycode As Integer, keystate As Long, shiftstate As
Integer, discard As Integer)
    discard = True ' get rid of the backslash
    If keystate And &H80000000 Then GoTo Bye: 'ignore KeyUp Event
    HookKbd1.Enabled = False ' accept only one message per keypress
    Select Case keycode
        Case 220            ' backslash
            SubClass1.PostEvent = 1' Send a posted message
    End Select
Bye:
End Sub
```

Now the time consuming code (highlighting the last character entered and copying it into the Clipboard) can be put safely into the SBC's DelayedEvent event.

```
Sub SubClass1_DelayedEvent (lvalue As Long)
    If (lvalue = 1) Then
        SendKeys "+{left 1}" 'select the last character typed
        SendKeys "^c"        'copy selected text to clipboard
        HookKbd1.Enabled = True    ' turn the SBCKBD back on
    End If
End Sub
```

Working with Caps-Lock During Keyboard Hooks

When using SpyWorks-VB's keyboard hook to detect hot keys that use the Shift key, you may need to consider the physical state of the Shift key, for example: when detecting cases where the user must hit the Shift key along with another key. Because the state of the CAPS LOCK key can cause keys to be shifted, special detection may be needed to see if the shift key itself is pressed.

For Example, if you want to have a "SHIFT-F10" key as a hot key, you have to detect both the Shift F10 and F10 keys with the keyboard hook. The trick is to call the GetKeyState API function to determine the state of the Caps Lock key during the KbdHook_Event. You can then do an exclusive or with SBCKBD's shiftstate parameter to determine whether the Shift key was actually hit.

This example shows how to detect for the physical state of the Shift key.

```
 capslockstate% = GetKeyState(VK_CAPITAL)
If (capslockstate% Xor shiftstate)
```

Then ... the shift key was physically pressed.

Keeping your Form TopMost

You can call the SetWindowPos API function to make your Form the topmost form (will always be ‘‘on top’’ of other windows and forms - unless others have also been set to topmost). But under certain situations, Windows and Forms can lose their topmost status (e.g. when minimizing a Visual Basic Form). A WM_WINDOWPOSCHANGING message is sent to the window that is losing their topmost status. You can use the SBC control to intercept that message and call the SetWindowPos API function to set the topmost status for that window again. SetWindowPos TopForm.hWnd,

Dear Marian.....

(Believe it or not, this column is not just a gimmick. The person responsible for producing the Newsletter is, in fact, named Marian, and she really does write this column based on questions that are frequently asked of us. Ed…).

We at Desaware are not afraid of hard questions - even those designed to make us squirm. Here are some of the toughest ones, along with the sometimes painful answers.

Will there be a 32 bit edition of SpyWorks?

Yes. It's hard to do because the operating systems (NT and Windows 95) are completely different, but we are making good progress. There will also be 32 bit OLE control based editions of VersionStamper-VB and the Custom Control Factory, and a 32 bit edition of the Common Dialog Toolkit.

When will the new programs be available?

The new programs depend on 32 bit OLE control technology, which means you need a 32 bit OLE control container to use them. Until we know when those containers will become available, we can't forecast a release date for our controls.

Will there be an updated edition of Dan's book?

You may have found it suddenly difficult to obtain a copy of the Visual Basic Programmer's Guide to the Windows API. No, it didn't go out of print, but there was a problem. You see, most technical books sell a large number of copies when they first come out, then gradually taper off, and the publisher plans accordingly. Things didn't quite work out that way with the VB Programmer's Guide. While it did taper off after the first several months, sales then began to increase and either held steady or grew through 5 printings. In September, it seems that sales jumped again and the entire fifth printing sold out in less than two months (it was supposed to last for at least four). That, combined with a paper shortage meant that the book was out of stock through much of October and November. On behalf of Ziff-Davis Press, we apologize for this miscalculation - it is now available and will remain in print for the foreseeable future.

Dan is currently working on a new book, the Visual Basic Programmer's Guide to the Win32 API. This book is an update to the current book, but is not a replacement. The intent is for the original book to continue to be the 16 bit API reference book, while the new book takes over on the 32 bit development side. Expect it sometime after Windows 95 is released. *(Well, technically it did show up sometime after Windows 95 was released - though perhaps a bit later than I had hoped. Ed.:-)*

Congratulations

Please join us in congratulating Franky Wong for being formally named as Vice President of Desaware. Franky has long been performing the duties commiserate with the position as everyone is well aware. Thanks for the helping hand and a job extremely well done.

The Desaware Visual Basic Bulletin

Volume 1 Issue 1

Welcome to our Newsletter

Welcome to the first edition of what we hope will be an ongoing newsletter. Why a newsletter, you ask? Because if we are going to impose upon you by sending a mailing, the least we can do is make it something worthwhile. It is true that you will find some product information in here (after all, we can't afford to send these out if they don't generate some sales), but you will also find tips and techniques that are useful to VB programmers in general, and of course, our customers in particular.
— Dan Appleman

Inside...

Product Information:

(Note: These are the original product descriptions as they appeared in the newsletter. For more detailed information, refer to the Desaware online catalog)

Transferring information between forms and applications using Windows messages.

Windows has long recognized the need for applications to exchange data with each other. Initially data could only be exchanged manually using the clipboard. Later, Microsoft implemented Dynamic Data Exchange (DDE) which allowed applications to send data to each other automatically. DDE is supported in Visual Basic, however it is relatively inefficient. Performance can be poor, and due to its asynchronous nature, timeout errors can occur.

An ideal solution to interform/ intertask communications is the use of Windows messages to transfer the data. It is easy for VB applications to send messages (See Using SendMessage on page 2), but a VB application requires the SBC.VBX control from Desaware's SpyWorks-VB package to be able to intercept incoming messages. Messages have the advantage of being both fast and synchronous (your code does not return from a SendMessage call until the other form or application has processed the message). A message triggers an event in the destination form that can include data or a pointer to the data, so you can avoid the need to use global variables.

The following steps are involved in using messaging to communicate between forms or applications.

Obtain a message number to use.

Obtain a destination window handle. For forms in your own application, you can simply use the Hwnd property for the form. For forms in other applications, use one of the techniques described later in this article to obtain a window handle..

The destination form uses **SBC.VBX** to subclass the form. This involves setting the ctlParam property to the name of the form to subclass, and setting the Messages or RegMessage property to identify the message to detect.

Prepare the data or obtain a handle or pointer to the data. (See: Using long variables to transfer data during callbacks and messaging, page 3).

Use the SendMessage API function to transfer the data.

On receipt of the message (SBC WndMessage event), the destination copies the data into the appropriate local variables.

Obtaining Message Numbers

Every Windows message is represented by a number. Standard Windows messages have constant values assigned, for example: the WM_PAINT message is always &H0F. If you need a message number to use within an application or between cooperating applications, you can also use a constant message number. The numbers above &H400 (WM_USER) are available for your use. You need not worry about other programs sending those messages to your form. For Visual Basic, you should avoid using message numbers between &H1000 and &H2FFF, as the lower part of these ranges are currently used internally by Visual Basic.

If you wish to obtain a message number that is unique in the system, you can use registered messages. Registered messages are obtained using the function **RegisterWindowMessage(lpstring$)**, where lpstring$ is the name of the message. Any application that calls **RegisterWindowMessage** with the same name will receive the same message number. This is ideal for cases where you wish to publish message numbers to use with other applications and be assured that they are unique.

Obtaining a Window Handle in Another Application

You must obtain a handle to a window in another application in order to send it a message. At first glance, the **FindWindow (classname$, windowname$)** API seems a suitable choice. Unfortunately, there are many cases where it is not reliable. Finding a window by class runs into problems any time more than one window of a class exists. For example: if more than one instance of NotePad is running, **FindWindow** will return the handle of the first one found, which may not be the one you want. The same problem applies to doing a **FindWindow** based on the window name (which is the same as the window caption), plus you have the added problem of handling those applications who modify their caption based on the open document. For example: the default windowname for Notepad is actually "Notepad - (Untitled)".

The real solution for this problem is to use the **EnumWindows** API function to enumerate all of the top level windows in the system, checking first if it belongs to the module handle that you are searching for (as returned by the Shell command or WinExec API) and also if it matches the class that you are looking for (for cases where an application has more than one top level window). The **EnumWindows** API function takes as one of its parameters a pointer to a function. Since Visual Basic does not provide you with an address of a VB function, you can use the **CBK.VBX** callback control (part of SpyWorks-VB) to obtain a function address to pass to **EnumWindows**. Each time that function is called, it will trigger a callback event in the **CBK** custom control.

Using the SendMessage API

The SendMessage API function is used to send Windows messages to other applications. Unfortunately, there is frequently confusion on how to handle the lParam parameter which is defined to be "As Any". The declarations that we provide in the *Visual Basic Programmer's Guide to the Windows API* are as follows:

```
Declare Function SendMessageBynum& Lib "User" Alias "SendMessage" (ByVal hWnd%, ByVal wMsg%, ByVal wParam%, ByVal lParam&)
Declare Function SendMessage& Lib "User" (ByVal hWnd%, ByVal wMsg%, ByVal wParam%, lParam As Any)
Declare Function SendMessageByString& Lib "User" Alias "SendMessage" (ByVal hWnd%, ByVal wMsg%, ByVal wParam%, ByVal lParam$)
```

We recommend that you use the aliased versions where possible to prevent errors, but here is a brief summary of how to send values, strings, structure addresses and nulls.

When the message expects a string:

```
dl& = SendMessageByString(hWnd, MSG, wParam, lParam$) or
dl& = SendMessage(hWnd, MSG, wParam, ByVal lParam$)
```

When a message expects a NULL value (can substitute a long variable as well)

```
dl& = SendMessageBynum(hWnd, MSG, wParam, 0) or
dl& = SendMessage(hWnd, MSG, wParam, ByVal 0&) (must be long type!)
```

When a message expects a pointer to a structure or numeric variable

```
dl& = SendMessage(hWnd, MSG, wParam,  lParam) (lParam is any variable or structure)
```

You can see how the aliased versions significantly reduce the chance of error, since the declaration takes care of both adding the ByVal and forcing the parameter to be passed as the correct type.

Using long variables to transfer data during callbacks and messaging

The **lParam** parameter to a window message will frequently point to a data structure. Many callback functions also use pointers. In some cases you simply need to examine values in the structure. In others you may need to change values in the structure.

The trick to doing this is to dimension an object of the referenced structure type in the event code itself. You can then use the **dwCopyData** function to copy the data to and from the structure.

For example: you can define a structure to transfer two integers and a string:

```
Type YourStruct
  A As Integer
  B As Integer
  S As String * 40
End Type
```

Sending the data via a message is easy. The source form or application uses SendMessage as follows:

```
Dim YS As YourStruct
SendMessage(desthwnd, msg, wp, YS)
```

This causes a pointer to structure YS to be passed as the lParam parameter. You can receive and process it as follows:

```
Sub WndMessage(wnd As Integer, msg As Integer, wp As Integer, lp As Long,
retval As Long, nodef As Integer)
  Dim YS2 As YourStruct
   ' Get a copy of the structure
  dwCopyData ByVal lp, YS2, Len(YS2)
  ‘ Now you can access the data using
  ‘ the fields in YS2
  x% = YS2.A
  YS2.S = “New string data”
  ‘ To set data into YS, copy the data
  ‘ back as follows
  dwCopyData YS, ByVal lp, Len(YS2)
End Sub
```

Be very careful using the dwCopyData function. You will always need to specify the **ByVal** keyword for lp. Be sure that the size of the structure is correct. It is also a good idea to check if lp is zero before attempting the copy.

The Newest Component: OLE custom controls

Microsoft has recently announced a new component model called OLE custom controls (though we expect everyone will soon be calling them by their extension - OCX controls). With all the press you may wonder, what is hype and what is real? Well, here's our opinion....

It has been apparent to us for some time that the VBX control model is limited and would not be easy to port to the 32 bit programming model. Thus in the long term there is no doubt that the VBX model will become obsolete. But then again, so will DOS, right? Yet how many people still use DOS applications?

As this paper goes to press, the next version of Visual Basic remains little more than rumor - it is clearly some time in the future. 32 bit application development is also clearly only in its earliest stages, though we expect it to accelerate with the appearance of Windows 4.0 sometime late this year (or so they say).

The VBX programming model is extensively supported by C++ compilers, application development platforms and Visual Basic. Microsoft has publicly said that VBXs will be supported under the 16 bit edition of the next edition of Visual Basic. We expect VBX to be the dominant component model for the rest of this year without any doubt, and probably well into 1995.

That said, we at Desaware are turning our attention *now* to the new OLE custom control format and do plan to support all of our products under this control model in both the 16 and 32 bit environments as quickly as possible. After all, our entire focus as a vendor is to provide the highest quality and most sophisticated developer's tools. So stay tuned!

Safely distribute your Visual Basic applications...

An end-user or customer calls. ***Your program is behaving strangely, or no longer loads***. Did another program install an older DLL or VBX on their system? Or maybe a later, but incompatible version? Is it a conflict due to an incompatible DLL already loaded in memory? Has the PATH environment setting changed? Is a required file missing? If only your program could tell you what's gone wrong... Now it can - with VersionStamper-VB

Detect VBX and DLL Incompatibilities Instantly

VersionStamper-VB allows you to embed into your Visual Basic executable information about all of the dynamic link libraries and custom controls used by that application, along with their required versions, file dates, and choice of warning conditions. The dwvstamp.vbx custom control uses this information to check the versions of the actual modules that are available on the target system. Incompatibilities are detected instantly.

VersionStamper-VB is designed with your needs in mind. You can perform version verification automatically on load, or at any time under program control. A customizable and redistributable "rescue" program can be used to detect and log problems when an application won't load at all. VersionStamper-VB is designed primarily for DLL's and VBX's, but will work with any type of file, even those without version information.

Simply set a few control properties in the dwvstamp.vbx custom control and VersionStamper-VB automatically adds a standard Windows version stamp into your Visual Basic based executable when you compile..

Add true Versioning to your Visual Basic EXEs

A COMPLETE solution

VersionStamper-VB is easy to use even if you are an absolute beginner - but we didn't forget the experts. Look what else you get:

dwvstamp.vbx - This version stamping custom control has properties for each of the standard version information fields, and extensive dialog box support for embedding version information for DLLs, VBXs and other files which you select.. It also has the ability to scan Visual Basic-based projects to automatically create a list of the DLLs and VBXs required by the application. The runtime distributable control is royalty free, and is small and efficient.

VerInfo.exe - A Visual Basic-based program that allows you to view the version information for any executable, DLL or VBX that contains versioning information. Source code is included. It also produces a complete list of the components required by the application based on information embedded by the dwvstamp.vbx control.

VerResQ.exe - A Visual Basic-based program that analyzes any Visual Basic-based executable stamped by VersionStamper-VB. It instantly detects any incompatibilities that may exist between the requested files and those actually present on disk. Source code is included so that it may be customized for distribution with your application.

VrsqTest.mak - A sample rescue program that shows how to scan a version stamped executable and determine any incompatibilities that exist - even if the application cannot be loaded. It is designed to be easily modified so that you can create your own rescue program to distribute with your application.

Technical Notes include a discussion of the Windows-based system of version stamping, using the versioning API, suggestions for improving the Visual Basic setup toolkit and more.

Whether you are developing for corporate use in house, or for distribution worldwide, VersionStamper-VB will pay for itself the first time a customer calls with a compatibility problem. You'll have a solution in minutes - with VersionStamper-VB.

NOTES:

VersionStamper-VB posed an interesting problem for us. How do you create a package that is at once so easy to use that even an absolute beginner can use it, yet powerful enough for the most advanced VB developer? We ended up placing 99.9% of the functionality into a single custom control and several

prepackaged forms and modules. Use the defaults and you have a complete solution using just a few lines of code. Or modify our VB code and override the VBX event defaults to create your own customized solution. Our manual is designed the same way: use the quick-start for a canned solution, or read the text (>100 pages) to learn everything you would want to know about versioning and component distribution.
- Dan

Distributing Applications Created with Visual Basic

Quite a bit of attention has been paid lately to the problem of source code control and group project management. Fortunately, a number of source code control projects have begun to appear. It is only recently that developers have turned their attention to the problem of safely distributing applications that make use of components such as Visual Basic custom controls (VBXs) and Dynamic link libraries. The presence of an obsolete component on a system can cause an application to fail, and even cause a General Protection Fault.

Developers have attempted a number of solutions. Some try to place their VBXs and DLLs in their own application directory. This approach can fail in two ways. First, once another application loads a different version of the control, that version will be used - Windows will not attempt to load the correct version. Next, it is not possible to guarantee that the current directory will be the same as the one containing the components, and Windows always searches the current directory first.

Some developers attempt to make sure that they have the correct control by installing their versions of a component into the system directory regardless of the presence of an existing control that may have a later version. This could almost be considered criminal, as it is very likely to break other applications on the end user's system.

Windows searches for components in the following order:

Any component with the same module name that is already in memory.

The current directory.

The Windows directory.

The System directory.

The project directory (contains the executable that is loading the component).

Directories specified by the PATH environment variable.

There is no way to completely assure that an incompatible control does not exist somewhere on a target system, or to prevent it from being loaded.

Worse yet, there is no way to determine the version of a Visual Basic executable, as VB does not support standard Windows version resources. This means that a user could take an older version of your application, and overwrite a later version without warning.

Given these facts, two things are clearly needed. First, a way to embed a version resource into a VB executable. Second, a way to mark an executable with a list of the components that it requires so that it would be possible to determine if the correct components are in fact present on a target system. Desaware's new VersionStamper-VB addresses both issues.

Make your VB Applications Tile like non-VB Applications

One of the biggest complaints about programs created with Visual Basic is that they cannot be tiled by the Windows Task Manager. Instead, Task Manager leaves the VB program where it was, and leaves a large hole where it was supposed to go. Desaware's SpyWorks-VB allows you to let your VB application tile with the rest.

First, you need to subclass the hidden main window that belongs to every VB application (class name ThunderRTMain - Use the **EnumTaskWindows** API with the **CBK** callback control to get this handle). This is because task manager tries to tile this window - not your form. But you can intercept the task manager's attempt to size this window and move the form yourself.

Use the **SBC.VBX** control to intercept the WM_WINDOWPOSCHANGING message and use the following code:

```
Sub SubClass1_WndMessage (Wnd As Integer, msg As Integer, wp As Integer, lp
As Long, retval As Long, nodef As Integer)
  Dim lpwinpos As Long, flags As Integer
  Dim isicon As Integer, ismax  As Integer
  Select Case msg
     Case WM_WINDOWPOSCHANGING:
           ' Make sure our application is not in iconic or
           ' maximized state before tiling or cascading.
           isicon = IsIconic(Me.hWnd)
           ismax = IsZoomed(Me.hWnd)
           If ismax Or isicon Then Exit Sub
            'get structure information
           lpwinpos = dwGetAddressForObject(winpos)
           dwCopyDataBynum lp, lpwinpos, Len(winpos)
           'The hwndInsertAfter field of the WINDOWPOS structure is
normally the handle of this apps  main form, but is NULL when a tile  or
cascade message is sent. The SWP_NOCOPYBITS bit of the flags field is turned
on during a tile or cascade message, and off for others.
           If (winpos.hwndInsertAfter = 0) And (winpos.flags And
SWP_NOCOPYBITS) Then
                flags = winpos.flags And &HFEFC
                ' Position and size the main form.
                SetWindowPos Me.hWnd, 0, winpos.x,  winpos.y, winpos.cx,
winpos.cy, flags
           End If
  End Select
End Sub
```

You also need to place code in the form's Resize event to hide the ThunderRTMain window when the form is minimized or maximized. Use the ShowWindow API to do this.

A sample program demonstrating VB tiling can be found on the Newsletter Bonus Disk described on page 7.

Trapping and blocking the Control-Escape Sequence

When the user hits the Ctrl+Esc key, Windows sends a WM_SYSCOMMAND message to the active application. The information in the message tells the active application to bring up the Task List. You cannot use the Keyboard Hook to throw the Ctrl-Esc key message away because the WM_SYSCOMMAND message is sent before the key message.

The solution is to use the **SBCHOOK.VBX** control to place a task or systemwide hook that detects the WM_SYSCOMMAND message (Set the HookType property to GetMessage).

In the WndMessage event, the lp parameter contains the cursor coordinate if the user selects the "Switch To..." command from the System menu. When the user hits the Ctrl+Esc key, lp is set to 0. By blocking the WM_SYSCOMMAND message only when lp = 0, you can allow the user to bring up the Task List via the System menu.

In the WndMessage event place the following code:

```
If msg = WM_SYSCOMMAND Then
   If wp = SC_TASKLIST Then
      If lp = 0 Then
           nodef = True
           msg = 0
     End If
  End If
End If
```

Setting nodef to True prevents further hooks from being called, but in many cases you must also set the msg value to zero to truly block the control-escape key.

Parsing Lines Read from Win.ini

What do you do when you have a list of words in a string that are separated by a delimiter such as a comma or null character, and you wish to find out if a word is in the list? You might use the Instr$ function, except that it also detects partial words such as "*now*" in "*snow*". You can use the Instr$ function however, if you include the delimiters in the search string. So in this case you would search for "*,now,*". The only problem with this approach is that it will fail to check the first and last word in the list because they only have one delimiter. The solution? Add an extra delimiter to the start and end of the list as follows:

Instr(",", list$ & "," , ",*now,*")

accurately detects the word "now" in the word list. This technique is especially useful when you use an initialization file that has a list of words separated by commas, or API functions that return a list of entries delimited by null characters.

NOTES:

When VB3 came out, I snagged a bunch of free copies of VB2 from a local user's group and handed them out to some local teens. One of them, Eyal Soha (16), learned enough to write the swiniedt program which became part of SpyWorks-VB, and came up with this clever tip in the process. — Dan

Handling Null Terminated Strings

Many Windows API functions can return a string to your application by loading a null terminated string into a buffer. The end of this string is marked by a null character, however Visual Basic will still use the entire length of the buffer in string operations. Consider this example:

Declare the **GetClassName** function that retrieves the Windows class name for a window:

```
Declare Function GetClassName% Lib "User" (ByVal hWnd%, ByVal lpClassName$,
ByVal nMaxCount%)
```

Then use this code to get a forms class:

```
classname$ = String$(32, "A")
di% = GetClassName(yourform.hWnd, classname$, 31)
```

The string buffer is filled with the letter A to illustrate clearly what is taking place.

Print classname$ in the immediate window:

You will see:

ThunderForm AAAAAAAAAAAAAAAAAAAA

Visual Basic allows embedded nulls, so even though the API correctly set the buffer to a null terminated string, the length of the string from VB's point of view is still 32, and the characters after the null terminator remain unchanged.

Fortunately, it is easy to extract a VB string that contains only that part of the buffer before the null termination:

```
Function NullTermToVBString$ (usestr$)
Dim position%
  position% = InStr(usestr, Chr$(0))
  If position% <= 1 Then ' Avoid errors
     NullTermToVBString$ = ""
     Exit Function
  End If
  NullTermToVBString$ = Left$(usestr$, position% - 1)
End Function
```

SpyWorks-VB - Winner of the Visual Basic Programmer's Journal 1994 Reader's Choice Award!

SpyWorks-VB allows you to do virtually anything in Visual Basic that is possible with other languages. It supports powerful Windows programming techniques that are not built into Visual Basic. Subclassing allows you to intercept any Windows message going to a form or control. Callbacks allow you to provide function pointers to Windows and other DLL's which trigger VB events when called. Hooks give you system or task access to keyboard and mouse events.

You CAN do it in Visual Basic

We can barely begin to describe all of the possibilities provided by SpyWorks-VB, but here are a few: Create VB add-ins to other applications or VB itself using cross-task subclassing. Printer configuration. MouseEnter/MouseExit for status bars. Tiny Captions and Rollup Windows. Scrolling Forms/Controls. True LostFocus & GotFocus detection. Create owner draw controls. Trap tab and alt-tab keys. Use Windows enumeration functions. Detect WIN.INI changes. Use private and owner draw clipboard formats. Change the behavior of standard controls. Indirect property access. Obtain an event history for a program. Detect Windows API parameter errors.

SpyWorks-VB 2.0 Custom Controls:

SBC.VBX - our subclassing custom control provides you complete access to the underlying windows message stream for the forms, controls and even graphical controls in your application. It even detects internal Visual Basic messages.

CBK.VBX - our generic callback custom control handles Windows API function or third party DLL functions that require a function address as a parameter. It even handles multimedia callbacks.

The **SBCEasy.VBX** custom control adds true Mouse enter and exit events, Menu selection events, "tiny" captions and "rollup" windows, virtual forms, and more.

The **SBCHOOK.VBX** custom control provides extensive support for Windows hooks allowing you to detect underlying Windows messages on a form, application or system level.

SBCKBD.VBX, the keyboard hook custom control, lets you detect all keystrokes for an application or the entire system. Even detects hard to get tab, enter and arrow keys.

The **DWSPYDLL.DLL** dynamic link library provides utility functions, access to Visual Basic CDK functions, and the ability to read and write property values for controls in other applications, and even the VB design environment.

SpyWorks-VB Debugging Tools

SpyParam.exe uses the Windows 3.1 built in parameter error detection capability to detect and trap API function parameter errors - errors that are hard to find because they are simply ignored by Windows.

SpyMem.exe is a memory, resource, module and task browser. Its snapshot mode allows you to compare current heap & resource states with a previous reference - ideal for detecting resource leaks and memory that is not being freed.

SpyMsg.exe is a classic "spy" program with the ability to record any or all messages going to one or more Windows in your system. It also has the unique ability to detect Visual Basic events.

SpyWin.exe is a window browser that lets you examine information about all of the windows in your system. Visual Basic awareness allows it to report VB specific information such as form and control names, and to detect graphical controls..

SpyMenu.exe is a menu browser that can analyze the structure of the menu and command identifiers for any window on your system.

SpyVBX.exe is a form, control and custom control analysis program.

SpyWorks-VB comes with extensive on-line help, lots of VB source examples, SpyNotes #1 (advanced application notes), and more....

Next time someone tells you that you can't do something in Visual Basic the joke will be on them. Now you CAN do it - with SpyWorks-VB.

(update - SpyWorks- just won the 1995 VBPJ Reader's Choice award as well)

NOTES:
I had the original vision for SpyWorks-VB within minutes of seeing the original VB 1.0 beta. The efficiency of VB was obvious, and once I saw that it was possible to access DLL's and create custom controls I knew that I would not have to compromise on power - I could still use the capabilities of Windows. It took over a year for this vision to evolve into a way to share this power with everyone - without requiring that they program in C or become Windows experts.
I'd like to take this opportunity to thank you all for your support and encouragement. You are the people who made SpyWorks the amazing success that it has become..
— Dan

SpyNotes #2 The Common Dialog Toolkit

The Common Dialog Toolkit, an applications pack for use with **SpyWorks-VB,** lets programmers access and modify the Windows Common Dialog Box library using API functions.

The common dialog box library included with Windows makes it possible for programmers to significantly customize the behavior and appearance of common dialogs. However, the common dialog custom control (VBX) that comes with Visual Basic Professional does not provide this capability. The Common Dialog Toolkit shows Visual Basic programmers how to directly access the Common Dialog libraries using the tools in SpyWorks-VB to take advantage of all of the capabilities of the Windows Common Dialogs.

Desaware has launched the **SpyNotes** series of application notes to help prog-rammers learn the advanced techniques made possible by SpyWorks-VB. The Common Dialog Toolkit is the second application note in this series. As such, in addition to a description of how to access Common Dialogs, the package includes in depth technical information and complete documented source code to do the following (and more):

How to access all of the standard Common Dialog boxes from VB.

Using callbacks to trap the messages received by Common Dialogs.

How to embed Visual Basic controls into a common dialog box.

How to use *modeless* dialog boxes and give *modal* dialogs *modeless behavior.*

How to add controls to a common dialog box.

How to set the size and position of a common dialog box.

How to register and use user-defined messages and registered messages.

How to create 3D common dialogs.

How to Hook into the main Visual Basic GetMessage loop.

How to access and modify the behavior of controls in a dialog box.

With its advanced technical information, the Common Dialog Toolkit will prove valuable even if you never use the Common Dialogs themselves!

Classics from Desaware - CCF-Cursors and The Custom Control Factory

Visual Basic comes with only 12 built in cursors (or "mousepointers" as they are called), and could not easily be extended. **CCF-Cursors** provides a complete package for design and use of custom cursors and includes other enhancements to VB's mouse and cursor control.

Create Cursors from scratch, or convert any Icon into a cursor.
Set the cursor for a form or control.
Animated cursors - automatic animation with specified intervals.
Support for Visual Basic 3.0 graphical controls.
Detect mouse events for ANY control.
Detect menu selection events.
Coordinate transformation, resource and other functions.
Over 50 sample cursors, including an animated hourglass.

The Custom Control Factory lets you create customized button controls interactively while working in VB design mode. With their unique ability to hold virtually unlimited numbers of bitmaps, icons, cursors or metafiles, CCF controls can be used to create **Animated Buttons**, **Multistate Buttons, Toolbars,** and other enhanced user interface objects.

Features include: Multiline Captions, flexible text/image alignment, image scaling and automatic control sizing, programmable animation speed, runtime configurable, automatic 3D borders with flexible widths and colors, 256 color support, Image compression, Custom drawing/cycles. Now includes our new **MLIST2.VBX** control, a powerful list box that also holds bitmaps and supports unique colors for each item in the list box. (Includes C source code for MLIST2).

How Computer Programming Works

A new book by Daniel Appleman
from Ziff-Davis Press.
ISBN: 1-56276-195-1

Have you ever tried to explain to someone what it is that you actually do as a programmer? Are you a self taught programmer who sometimes wonders if maybe you missed some important information along the way - or do you know someone in that situation? Have you ever met someone, a friend, parent or child that you wanted to introduce to programming in an easy to read manner?
Well... I've been in all of these situations - and I have wanted to do something about it for a long time. Now, thanks to the good folks at Ziff-Davis Press, I've been able to take what I think it is a good step in the right direction.
''How Computer Programming Works'' is part of Ziff's award winning How It Works series. It's fully illustrated (color, of course) and focuses on the fundamentals and concepts of programming and computer science. Plus, I've arranged again to offer a 20% discount on the book when purchased through Desaware.
— Dan

Visual Basic Programmer's Guide to the Windows API

Build on your knowledge of Visual Basic to become a Windows expert by learning to take advantage of the hundreds of functions contained in the Windows API. No knowledge of 'C' or prior Windows experience is required. It covers virtually every aspect of Windows and includes extensive sample code. Even Windows experts will benefit from the information on API/VB compatibility. Over 1000 pages plus disk. Published by Ziff-Davis Press. ISBN: 1-56276-073-4. 20% discount on this book when purchased through Desaware.